6 KAMPOW
SZCZEPANOWSKI

MICRO APPLICATION

ATARI ST

LE LIVRE DU ST BASIC

UN LIVRE DATA BECKER

6

KAMPOW
SZCZEPANOWSKI

MICRO APPLICATION

ATARI ST

LE LIVRE DU ST BASIC

UN LIVRE DATA BECKER

Distribué par : MICRO APPLICATION
13, Rue Sainte Cécile
75009 PARIS

et

EDITION RADIO
9, Rue Jacob
75006 PARIS

(c) Reproduction interdite sans l'autorisation de
MICRO APPLICATION

'Toute représentation ou reproduction, intégrale ou partielle, faite sans le consentement de MICRO APPLICATION est illicite (Loi du 11 Mars 1957, article 40, 1er alinéa).

Cette représentation ou reproduction illicite, par quelque procédé que ce soit, constituerait une contrefaçon sanctionnée par les articles 425 et suivants de Code Pénal.

La Loi du 11 Mars 1957 n'autorise, aux termes des alinéas 2 et 3 de l'article 41, que les copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à l'utilisation collective d'une part, et d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration'.

ISBN : 2-86899-045-8

(c) 1985 DATA BECKER
Merowingerstrasse, 30
4000 DUSSELDORF
R.F.A.

Traduction Française assurée par Pascal HAUSMAN

(c) 1985 MICRO APPLICATION
13 Rue Sainte Cécile
75009 PARIS

Collection dirigée par Mr Philippe OLIVIER
Edition réalisée par Frédérique BEAUDONNET

TABLE DES MATIERES

INTRODUCTION 1

1. LES BASES DE LA PROGRAMMATION 3

1.1 Explications générales 3

1.2 Le langage de programmation BASIC 4

1.3 Flux de données et organigrammes 6

1.3.1 Flux de données 8
1.3.2 Les organigrammes 12

1.4 Codes ASCII 17

1.5 Les systémes numériques 18

1.5.1 Le système binaire 18
1.5.2 Bit et octet 20
1.5.3 Le système hexadécimal 20

1.6 Les opérateurs logiques 23

1.6.1 NOT 24
1.6.2 AND 25
1.6.3 OR 26
1.6.4 XOR 28
1.6.5 EQV 30
1.6.6 IMP 30

Exercices 31

2. INTRODUCTION A LA PROGRAMMATION EN BASIC 32

2.1 Le premier programme BASIC 32

2.1.1 Entrée de valeurs avec INPUT 36
2.1.2 Affectation d'une valeur avec LET 37
2.1.3 Sortie avec PRINT 39
2.1.3.1 PRINT USING 42
2.1.4 Commentaires avec REM 49

2.2 Les variables et leur utilisation dans des programmes 50

2.2.1 Opérations de calcul avec des variables 52

Exercices 54

2.3 Fonctions numériques 55

2.3.1 Fonctions avec DEF FN 60
2.3.2 Nombres aléatoires 61
2.3.3 Autres instructions pour manier les variables 63
2.3.4 ASC(X$) et CHR$(X) 65

Exercices 67

2.4 TAB et SPC 68

2.5 Les chaînes de caractères 69

2.5.1 LEFT$ 70
2.5.2 RIGHT$ 72
2.5.3 MID$ 73
2.5.4 LEN(X$) 75
2.5.5 VAL(X$) 75

2.5.6 STR$(X) 77
2.5.7 INSTR 78
2.5.8 STRING$ 79
2.5.9 SPACE$ 80

Exercices 81

2.6 L'édition des programmes 83

2.7 Les fenêtres écran 89

3. STRUCTURES DE PROGRAMME PLUS COMPLEXES 91

3.1 Saut inconditionnels 91

3.2 Sauts conditionnels 95

3.2.1 IF...THEN...ELSE 95
3.2.2 Etiquettes de saut / labels 99

Exercices 101

3.2.3 FOR...TO...NEXT 102
3.2.4 WHILE...WEND 109

3.3 Instructions de sauts calculés 114

3.3.1 Programme d'exemple "cours de calcul" 118
3.3.2 Sauts avec ON ERROR GOTO 127

Exercices 130

3.4 L'interrogation du clavier 131

3.5 FRE, POS, CALL, WAIT 132

3.6 READ, DATA et RESTORE 134

4. APPLICATIONS BASIC PLUS COMPLEXES 140

4.1 Les tableaux 140

4.1.1 Les tableaux à une dimension 140
4.1.2 Exemples d'applications des tableaux unidimensionnels 148

Exercices 157

4.1.3 Les tableaux à plusieurs dimensions 158

4.2 Sous-programmes 168

4.3 La technique des menus 189

4.3.1 Utilisation de routines d'entrée dans le Menu 193
4.3.2 Positionnement du curseur avec GOTOXY 199

4.4 Procédures de tri 200

5. LE TRAVAIL AVEC LE LECTEUR DE DISQUETTE 204

5.1 La gestion des programmes 204

5.1.1 Sauvegarde de programmes 204
5.1.2 Chargement de programmes 206
5.1.3 Affichage du contenu de la disquette 206
5.1.4 Suppression de fichiers 207
5.1.5 Changement du nom de fichiers 208

5.2 Gestion de fichiers séquentiels 208

6. MUSIQUE ET GRAPHISME 213

6.1 La musique 213

6.2 Le graphisme 217

6.2.1 Lignes 218
6.2.2 Cercles 219
6.2.3 Ellipses 219
6.2.4 Surfaces remplies 220

7. FONCTIONS GEM SOUS BASIC 222

7.1 Les bases de GEM 222

7.2 Transmission de paramètres à des routines GEM 224

7.2.1 Appels de VDI 225
7.2.2 Appels d' AES 225

7.3 Exemples VDI 227

7.4 Exemples AES 228

8. SOLUTION DES EXERCICES 231

ANNEXE

LISTE D'INSTRUCTIONS 246

INTRODUCTION

Vous avez acquis, avec l'ATARI ST, un langage BASIC extrêmement puissant. Si la documentation livrée avec le BASIC ST était d'une qualité égale à celle de l'interpréteur, rien ne vous empêcherait de réaliser aisément des programmes puissants.
Le présent ouvrage n'a pas seulement pour but de combler cette lacune mais également de vous inculquer des connaissances de base sur le BASIC, qui vont de la simple sortie sur écran avec PRINT jusqu'à des algorithmes de programmation plus complexes comme, par exemple, le tri. En outre, cet ouvrage attirera votre attention sur les erreurs qui affectent la version actuelle du BASIC.

La première partie de cet ouvrage traite des bases générales de la programmation.
Comment arriver à un bon style de programmation ?
Comment commenter un programme de façon claire ?
Nous répondrons à ces questions et nous vous donnerons également quelques notions de base sur le traitement des données.

Les deuxième et troisième parties seront consacrées au travail de programmation proprement dit. Nous vous montrerons tout d'abord avec de nombreux exemples comment les diverses instructions BASIC peuvent être employées et à quoi elles servent. Nos exemples de programmes peuvent d'ailleurs tourner - à peu d'exceptions près - sur d'autres ordinateurs disposant du même jeu d'instructions. Nous avons en effet évité d'utiliser dans nos programmes trop de PEEK et de POKE car ces instructions permettent d'obtenir des résultats qui dépendent directement des adresses spécifiques de chaque ordinateur. L'utilisation de ces instructions produit donc des effets fort différents suivant l'ordinateur employé.

Vous trouverez à la suite des différents chapitres des exercices que nous vous invitons vivement à essayer de résoudre. Ils vous permettront en effet de vérifier que vous avez bien suivi nos développements jusqu'ici.

Vous trouverez les solutions de ces exercices, largement expliquées, dans un chapitre spécial, à la fin de ce livre. Etudiez les solutions des exercices d'un chapitre avant de passer au suivant car les solutions proposées font toujours un lien avec le chapitre suivant.

La quatrième partie traite de problèmes plus complexes auxquels répondent aussi des programmes plus complexes. Nous essaierons de vous faire maîtriser de tels problèmes et nous vous fournirons ici aussi de nombreux exemples et des exercices avec leurs solutions, de sorte à ce que vous n'ayez pas seulement à faire une lecture théorique mais que vous puissiez effectivement vous entraîner.

La cinquième partie est une introduction au principe de la gestion de fichier et du maniement du lecteur de disquette.

La sixième partie vous explique, par des exemples, quelques instructions graphiques et musicales de l'Atari ST.

La septième partie traite de l'appel de fonctions GEM à l'intérieur de programmes BASIC. Là encore, vous trouverez à nouveau quelques exemples.

Il est à noter que cet ouvrage est fondé sur la version BASIC du 18-7-85 (138944 octets). Il est donc possible qu'apparaissent quelques divergences entre certaines de nos explications et des versions qui pourraient apparaître ultérieurement.

Il ne nous reste plus maintenant qu'à vous souhaiter beaucoup de plaisir et de réussite dans votre travail avec cet ouvrage. Surtout ne vous désespérez pas si vous rencontrez de temps à autre une petite difficulté. Personne n'a encore réussi à s'improviser programmeur du jour au lendemain et surtout, le travail de programmation exige qu'on puisse avoir une certaine patience dans la recherche des erreurs.

Les auteurs

CHAPITRE 1

LES BASES DE LA PROGRAMMATION

1.1 EXPLICATIONS GENERALES

Nous allons traiter dans ce chapitre essentiellement des bases de la programmation. Avant que nous ne vous montrions, avec des exemples simples puis plus complexes, ce qu'est la programmation en BASIC, nous allons nous attacher ici à des notions plus générales, c'est-à-dire que nous allons expliquer comment un problème peut être transformé en un programme. Ces petits développements théoriques pourront vous sembler peu attrayants mais ils n'en sont pas moins très utiles et ils vous aideront à aborder plus tard des programmes plus complexes.

Qu'est-ce au juste que la programmation?

Vous devez partir du principe que l'ordinateur que vous venez d'allumer est parfaitement "stupide". Certains ordinateurs disposent d'un langage de programmation intégré mais, sur l'Atari ST, ce langage doit d'abord être chargé d'une disquette dans l'ordinateur. Malgré tout, si vous tapez simplement sur le clavier :

"calcule le volume d'une sphère de 2 mètres de rayon",

il ne faut pas vous attendre à ce que l'ordinateur vous réponde autre chose que "syntax error". Si vous voulez en effet faire résoudre un problème de cet ordre par votre ordinateur, vous devez indiquer à celui-ci ce qu'il doit faire non pas en français mais dans son langage propre. Vous devez donc écrire dans un ordre logique la liste des tâches qu'il devra accomplir pour trouver une solution à votre problème avec les moyens dont il dispose. La marche à suivre que vous définissez de cette manière s'appelle un "algorithme".

La suite d'instructions dont il se compose constitue un "programme".

1.2 LE LANGAGE DE PROGRAMMATION BASIC

Le langage auquel nous allons nous intéresser est, dans le cas de votre Atari ST, le BASIC. Le BASIC a été développé en 1961 au Darthmouth College dans le New Hampshire (USA) et son nom est une abréviation de "BEGINNER'S ALL purpose SYMBOLIC INSTRUCTION CODE", ce qui signifie "code d'instructions symboliques polyvalent pour débutants".
Le BASIC a été développé à partir du langage de programmation FORTRAN. Depuis sont apparus sur les différents ordinateurs des dialectes du BASIC, de sorte que le BASIC de l'Atari ST par exemple ne peut être utilisé directement sur d'autres ordinateurs. Les différences entre les différents dialectes BASIC ne concernent le plus souvent que des points de détail mais il est important de savoir que les programmes réalisés avec la version Atari ST du BASIC ne pourront parfois tourner sur d'autres ordinateurs qu'après quelques adaptations.

L'ordinateur ne comprend pas directement les instructions BASIC. Il faut en effet que celles-ci soient d'abord traduites dans le langage de la machine, appelé langage-machine, qui est le seul langage de programmation que l'ordinateur peut comprendre directement. La traduction des instructions BASIC est effectuée par l'interpréteur BASIC de l'ordinateur. Si vous entrez une instruction BASIC au clavier et que vous appuyez sur la touche RETURN, cette instruction est d'abord lue par l'interpréteur, puis traduite en code machine propre à l'ordinateur et ensuite seulement exécutée.

Nous dirons donc pour nous résumer que la programmation consiste à traduire un algorithme dans un langage de programmation, le langage qui nous intéresse étant le BASIC.

Voici comment les débutants, mais aussi beaucoup de programmeurs plus expérimentés, procèdent :

Monsieur Durand souhaite faire calculer le volume d'une sphère avec 20 rayons différents.
Il lit la formule dans son livre de géométrie : $V=4PIr^3/3$ et il la "rentre" dans l'ordinateur telle quelle :

```
10 PI=3.14159265
20 FOR I=1 TO 20
30 INPUT"QUEL RAYON (EN CM)";R
40 V=4*PI*R^3/3
50 PRINT"LE VOLUME EST DE: ";V;" cm3"
60 NEXT I
```

Le programme tourne parfaitement et monsieur Durand obtient les résultats des calculs dont il avait besoin. Que peut-il vouloir de plus ? Il a trouvé un algorithme correspondant au problème qu'il voulait résoudre et a traduit cet algorithme en BASIC. Des programmes aussi simples donnent toujours la tentation de procéder de cette façon.

Mais dès que les problèmes à résoudre deviennent plus complexes, vous ne pouvez plus saisir d'un seul coup d'oeil le flux de données et le déroulement du programme. Il peut ainsi arriver qu'un programme tourne de façon incorrecte et que vous récoltiez ainsi des résultats erronnés. Cela provient alors du fait que vous avez certainement fait une erreur de logique dans votre programme et que celui-ci ne tourne pas comme vous le vouliez. L'être humain a en effet en général du mal à penser "à la place" de l'ordinateur et à anticiper parfaitement toutes ses réactions. Pour que l'ordinateur puisse résoudre un problème, celui-ci doit être décomposé en de nombreuses étapes élémentaires que l'ordinateur pourra suivre dans l'ordre où elles se présentent. C'est justement le fait de décomposer un problème et d'ordonner correctement les diverses étapes à suivre qui provoque toujours chez les programmeurs des erreurs plus ou moins nombreuses.

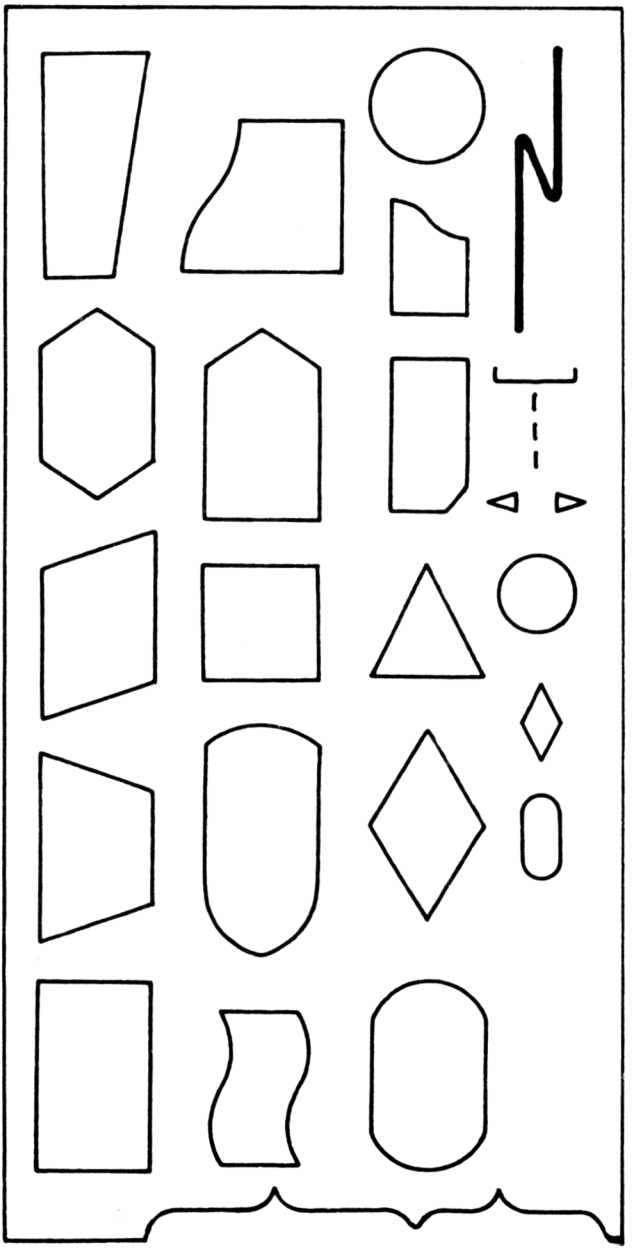
Le chemin qui mène de la définition d'une tâche à un programme achevé est donc plus complexe qu'il n'y paraît au premier abord. C'est pourquoi on introduit une étape intermédiaire dans laquelle on détermine ce que l'ordinateur devra faire, et dans quel ordre il devra le faire.

1.3 FLUX DE DONNEES ET PLANS DE DEROULEMENT DU PROGRAMME

L'étape intermédiaire à laquelle nous avons fait allusion sera décrite dans cette section. Il s'agit de créer des organigrammes, flux de données et plans de déroulement du programme d'après la norme DIN 66001. Le présent chapitre constitue une brève introduction à cette technique.

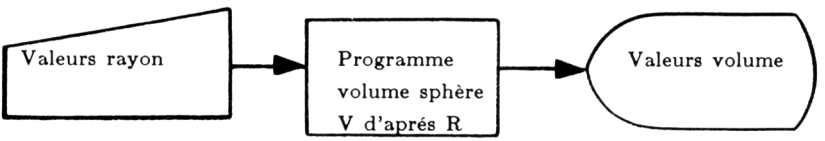
Les symboles utilisés pour réaliser des organigrammes figurent sur des plaques normographes que vous pouvez trouver dans le commerce. Voici l'illustration d'une telle plaque :

SYMBOLES POUR ORGANIGRAMMES



1.3.1 PLANS DE FLUX DE DONNEES

Les plans de flux de données doivent montrer quelles données (par exemple le rayon d'une sphère) doivent être entrées dans l'ordinateur et comment (par exemple manuellement à l'aide du clavier), par quels programmes les données seront traitées (par exemple programme de calcul du volume d'une sphère) et comment les données traitées seront ressorties (par exemple sur l'écran). Voici donc maintenant un exemple de plan de flux de données pour le programme calculant le volume d'une sphère en fonction du rayon entré manuellement :



Entrée au clavier **Traitement** **Affichage à l'écran**

FIGURE 2

Vous voyez qu'on peut réaliser un plan de flux de données même pour un programme aussi simple. Cela peut vous sembler ridicule mais il faut bien comprendre que la réalisation d'organigrammes sur des petits programmes n'a pas pour objet premier la réalisation de ces petits programmes mais plutôt l'apprentissage de la technique des organigrammes qui, lorsqu'elle vous sera devenue familière à force d'exercices aussi simples que celui-ci, vous rendra de grands services pour vous attaquer à des programmes beaucoup plus complexes où la réalisation d'organigrammes devient complètement indispensable. Les organigrammes pour de gros programmes peuvent fort bien occuper plusieurs pages.

Il faut en effet reconnaître que sur les listings de très gros programmes il est extrêmement difficile de suivre le traitement des différentes données et que, pratiquement, seul le programmeur auteur du programme y arrive (souvent péniblement). Si vous vous habituez dès le départ à travailler avec des plans de flux de données, vous aurez beaucoup moins de mal à en réaliser lorsque cela deviendra absolument indispensable à cause de la taille du programme.

La figure 3 explique la signification des différents symboles de plans de flux de données.

Avant de poursuivre la lecture de ce chapitre, nous vous proposons d'essayer de réaliser un plan de flux de données pour un programme convertissant les milles en kilomètres et affichant le résultat à l'écran. Comparez le plan de flux de données que vous avez réalisé avec la solution proposée par la figure 6.

Comme nous l'avons vu, les plans de flux de données permettent de représenter quelles données arriveront dans l'ordinateur à travers quels récepteurs de données, par quels programmes ces données seront traitées pour obtenir d'autres données et sur quels récepteurs de données ces nouvelles données seront sorties.

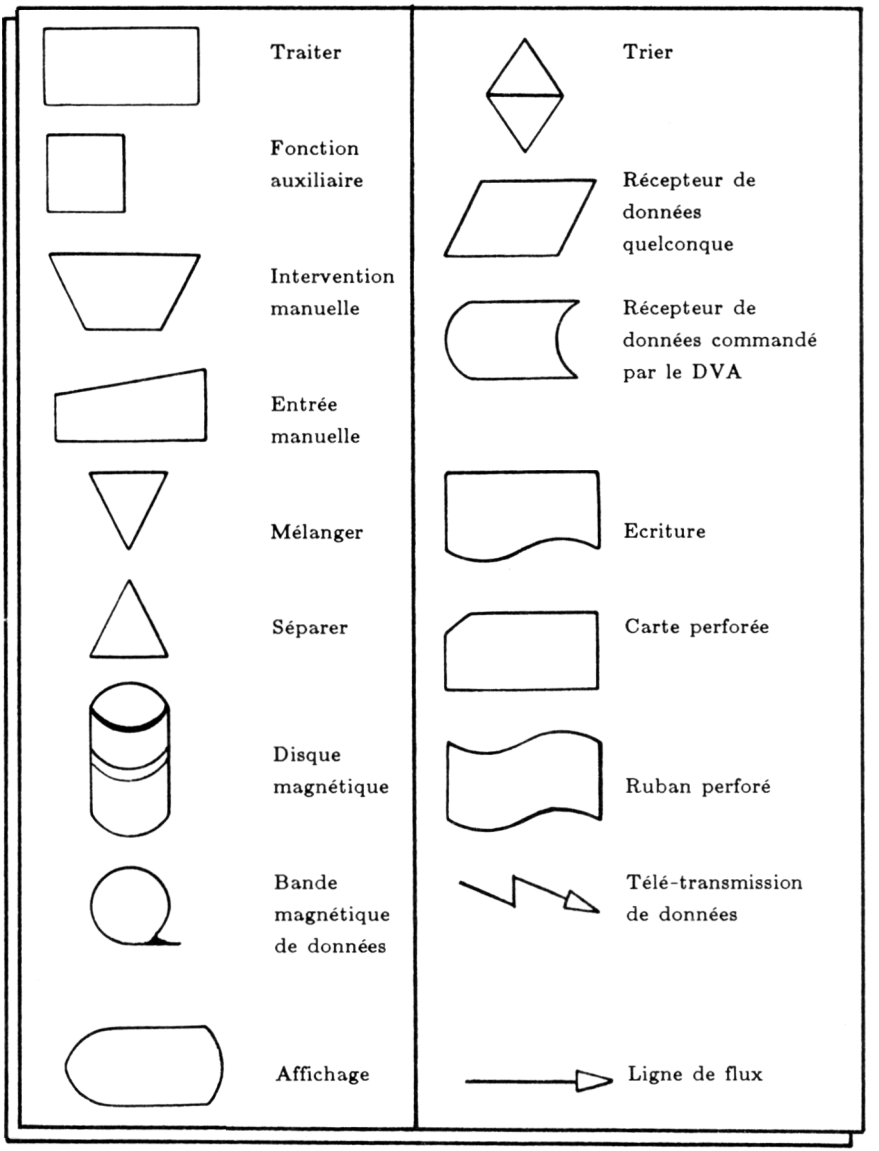

Traiter
Fonction auxiliaire
Intervention manuelle
Entrée manuelle
Mélanger
Séparer
Disque magnétique
Bande magnétique de données
Affichage
Trier
Récepteur de données quelconque
Récepteur de données commandé par le DVA
Ecriture
Carte perforée
Ruban perforé
Télé-transmission de données
Ligne de flux

FIGURE 3

Nous allons maintenant en venir au deuxième stade de l'étape intermédiaire qui est l'objet de ce chapitre, le plan de déroulement du programme ou organigramme proprement dit. En effet nos plans de flux de données ne nous disent pas par exemple comment le rayon d'une sphère va être traité pour obtenir le volume de cette sphère. Nous avons donc besoin d'une seconde forme de représentation symbolique décrivant toutes les étapes suivies par l'ordinateur pour résoudre un problème.

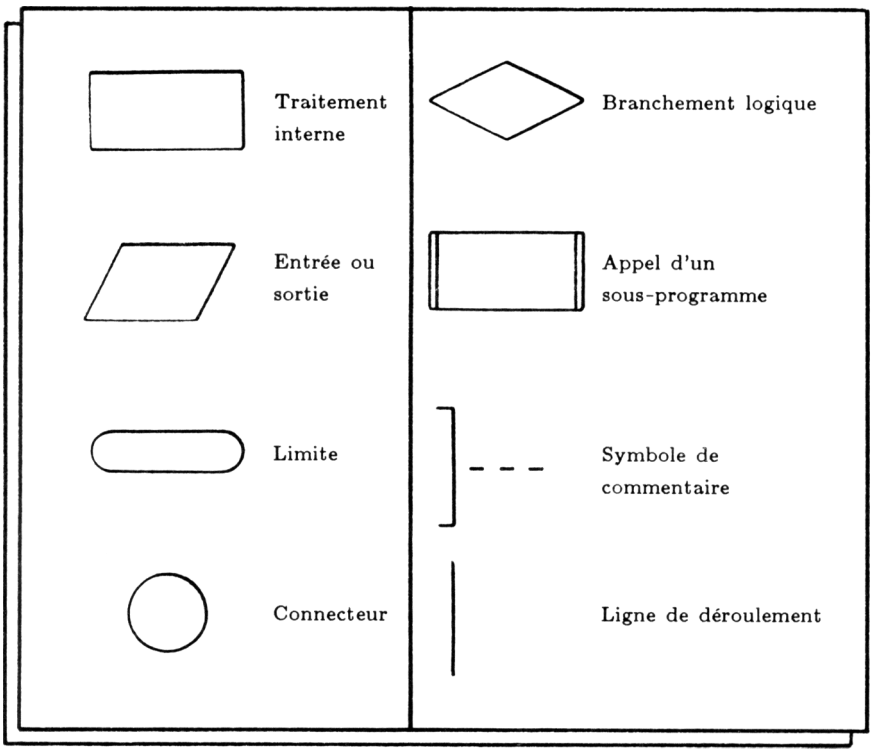


FIGURE 4

1.3.2 LES ORGANIGRAMMES

Le plan de flux de données pour le calcul du volume d'une sphère indique sous le point "traitement" uniquement "programme volume d'une sphère, V d'aprés R". Ceci indique donc seulement ce que deviennent les données entrées. Le problème lui-même n'a pas été décomposé en différentes étapes de traitement. C'est ce que doit réaliser l'organigramme. Un organigramme doit permettre de suivre les différentes étapes de la résolution d'un problème par un ordinateur. On utilise ici aussi les symboles conformes à la norme DIN 66001 que vous pouvez également trouver sur une plaque normographe. La signification des différents symboles est expliquée à la figure 4.

Nous allons maintenant essayer de réaliser l'organigramme correspondant toujours à notre même exemple. Comme nous l'avons déjà fait pour les plans de flux des données, nous ne saurions trop vous recommander de recourir systématiquement aux organigrammes, même pour des programmes pour lesquels cela ne s'impose pas, de façon à ce que vous maîtrisiez pleinement cette technique le jour où vous aurez à l'appliquer à des problèmes et donc des programmes beaucoup plus longs et compliqués.

Les organigrammes se dessinent toujours de haut en bas. Si vous arrivez au bas de la page, la partie suivante doit être dessinée à la droite du début de votre organigramme. Ne prenez surtout pas la mauvaise habitude de relier par une ligne deux parties d'un organigramme mais utilisez plutôt le symbole connecteur (voir la figure 4). Ce symbole doit être placé en bas de la première partie de votre organigramme et marqué d'un chiffre ou d'une lettre. Il suffit ensuite de placer un second connecteur, marqué de la même façon, au début de la deuxième partie de votre organigramme. Nous vous invitons maintenant à examiner l'organigramme que vous présente la figure 5. Les symboles de début ou de fin n'ont pas à être traduits en BASIC. Le symbole "ENTREE RAYON" peut être traduit par l'instruction BASIC INPUT. Cette instruction peut en outre recevoir un commentaire tel que "QUEL RAYON EN CM".

La formule de calcul du volume de la sphère peut être placée directement dans le symbole pour le traitement interne des données. Le symbole de sortie "sortie du volume" peut être traduit par une instruction PRINT suivie du texte correspondant. Au contraire de ce qui était le cas de notre programme précédent, nous n'avons pas utilisé ici de boucle FOR NEXT. Vous voyez que, lorsque l'organigramme a atteint un degré suffisant de précision, les différents symboles n'ont plus qu'à être remplacés par les instructions correspondantes du langage de programmation utilisé.

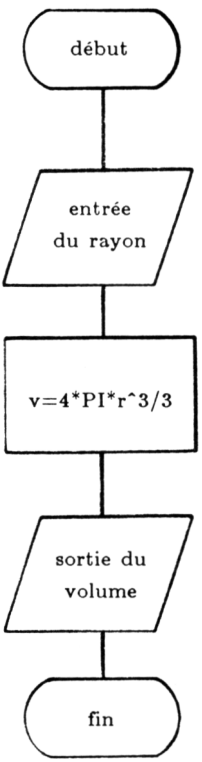


FIGURE 5

Une fois que vous en êtes arrivé à ce stade de la programmation, vous pouvez tenter un premier test de déroulement de votre programme. Ce test doit d'abord être effectué sur le papier, c'est-à-dire que vous devez suivre encore une fois le chemin des données sur votre plan de flux des données et contrôler le déroulement du programme tel qu'il est représenté par l'organigramme. Si tout vous semble correct, vous pouvez entrer votre programme en BASIC et le lancer avec l'instruction RUN.

Essayez maintenant de dessiner votre propre organigramme pour résoudre le problème suivant : convertir les températures exprimées en degrés Celsius en degrés Fahrenheit.

La formule est : F = 1.8*C+32

Nous espérons que vous trouverez la solution assez rapidement. Comparez votre organigramme à la figure 7.

Bien sûr c'est seulement quand vous aborderez des programmes plus importants que vous prendrez pleinement conscience des avantages de la technique des organigrammes. Ceux-ci peuvent en effet être examinés et compris rapidement du fait de leurs caractères graphiques, alors qu'on ne peut tout de même pas en dire autant des longs listings de programmes BASIC. Un autre avantage des organigrammes qu'il convient de ne pas perdre de vue est que ceux-ci sont indépendants de l'ordinateur utilisé. Enfin, les organigrammes constituent un moyen privilégié pour documenter un programme.

Insistons quelque peu sur la documentation des programmes qui est négligée de façon réellement inexcusable par de nombreux programmeurs. Il faut bien se dire une fois pour toute qu'aucun programmeur ne peut garder en tête indéfiniment la structure logique d'un programme. Si vous ne documentez pas vos programmes, cela veut dire que pour apporter des modifications à un programme auquel vous n'avez plus touché depuis plusieurs semaines, vous risquez de devoir tout d'abord lire et relire le listing de façon à "comprendre" comment votre programme fonctionne.

Ce n'est qu'après une étude longue et ennuyeuse de votre oeuvre que vous pourrez à nouveau la modifier.

Nous n'en dirons pas plus sur les plans de flux de données et les organigrammes et nous vous renvoyons si vous souhaitez approfondir cette matière à la littérature spécialisée dans ces questions.

Résumons une dernière fois les étapes que devrait normalement parcourir le travail de programmation :

1. *Définition du problème* (travail sur la définition du problème, analyse du problème).

2. *Projet d'algorithme de solution (plan de flux de données et organigramme ou plan de déroulement du programme).*

3. *Traduction de l'algorithme dans un langage de programmation* (réalisation du programme).

4. *Test du programme.*

5. *Documentation du programme.*

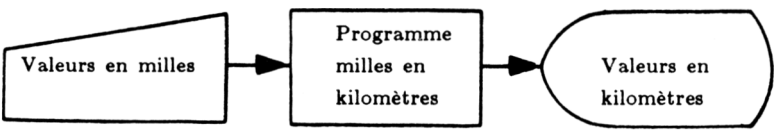


FIGURE 6

PROPOSITION DE SOLUTION

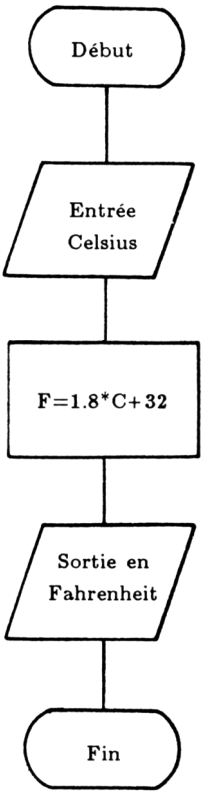


FIGURE 7

1.4 CODES ASCII

L'Atari ST ne peut pas traiter directement les caractères (les lettres, chiffres et autres signes) que vous entrez avec le clavier. Ces caractères correspondant aux touches sur lesquelles vous appuyez doivent être traduits dans un code numérique. Le code numérique le plus répandu est le **code ASCII**. ASCII signifie "American Standard Code for Information Interchange" ce qui signifie "Code standard américain pour l'échange d'informations".

Ce standard a été créé de façon à rendre possible des échanges d'informations même entre des ordinateurs différents. Par exemple, le caractère "A" correspondra toujours à la valeur 65 sur les ordinateurs ayant adopté le standard ASCII. L'éloignement entre émetteur et récepteur d'informations ne joue, à cet égard, aucun rôle. Que vous entriez des caractères au clavier - ils sont alors également envoyés à l'ordinateur à travers un câble - ou que vous envoyiez vos données aux Etats-Unis à travers un modem de télécommunication, dès que le récepteur recevra la valeur 65, celle-ci sera traduite en un "A". Le standard ASCII utilise des valeurs allant de 0 à 127.

La plupart des constructeurs d'ordinateurs ont cependant recours à un code ASCII étendu, de façon à ce que certains caractères puissent être également représentés en fonction des souhaits du constructeur ou de l'utilisateur. Ce code est également appelé code ASCII bien qu'il ne soit pas identique au code standard ASCII pour toutes les valeurs.

Le code standard ASCII réserve les valeurs 0 à 31 pour les caractères de commande, les valeurs 32-90 pour les majuscules et les valeurs 91-127 pour les minuscules et quelques autres signes. Le code ASCII de l'Atari ST ne correspond à ce standard que pour les codes de 32 à 91 ainsi que pour un petit nombre d'autres codes.

1.5 LES SYSTEMES NUMERIQUES

L'ordinateur ne peut distinguer que deux états dans ces circuits électroniques : allumé et éteint. Il a donc fallu créer un système numérique adapté à cette contrainte technique, le système binaire. Le système binaire ne comporte que 2 chiffres 0 et 1, alors que le système décimal qui nous est le plus familier en comporte 10. Le 1 du système binaire correspond à l'état "allumé" des circuits électroniques et le 0 à l'état "éteint". Pour expliquer le système binaire, partons du système décimal pour essayer de dégager les caractéristiques générales des systèmes numériques qui s'appliquent donc aussi bien au système binaire qu'au système décimal. En système décimal, nous pourrions représenter le nombre 5768 également sous la forme suivante :

$$5768 = 5*1000 + 7*100 + 6*10 + 8*1$$

ou encore :

$$5768 = 5*10^3 + 7*10^2 + 6*10^1 + 8*10^0$$

Souvenez-vous qu'en mathématiques, tout nombre élevé à la puissance 0 est égal à 1.

Dans le système décimal, les nombres sont donc représentés par une somme de produits en base 10. Chaque chiffre régit une puissance de 10 donnée :

10^3	10^2	10^1	10^0
5	6	7	8

1.5.1 LE SYSTEME BINAIRE

Le système binaire répond au même principe mais sa base est 2 au lieu de 10.

C'est pourquoi ce système ne connait que deux chiffres, 0 et 1. Pour convertir en un nombre décimal le nombre binaire 1011, il nous faut procéder ainsi :

Chaque chiffre d'un nombre binaire régit une puissance de 2. Pour convertir un nombre binaire, il suffit donc d'écrire chaque chiffre en dessous de la puissance de 2 correspondante. Il ne reste plus qu'à additionner entre elles les valeurs ainsi obtenues pour avoir le nombre décimal équivalent.

2^3	2^2	2^1	2^0
1	0	1	1

Additionnons ces valeurs :

$1*2^3 + 0*2^2 + 1*2^1 + 1*2^0 = 11$

ou

$1*8 + 0*4 + 1*2 + 1*1 = 11$

Le résultat est le nombre décimal 11. Pour convertir un nombre décimal en un nombre binaire, il faut procéder ainsi :

Pour convertir par exemple le nombre décimal 167 en un nombre binaire, il faut tout d'abord rechercher quelle est la plus grande puissance de 2 inférieure ou égale à ce nombre. Dans notre exemple, 2^8 (=256) est supérieur à 167 mais 2^7 (=128) est encore inférieur à 167. Cette valeur doit maintenant être soustraite du nombre décimal à convertir. Il faut ensuite procéder de même avec le reste 39. 2^6 (=64) est supérieur à 39 mais 2^5 (=32) est inférieur à 39. Reste 7. 2^4 (=16) et 2^3 (=8) sont supérieurs à 7 mais 2^2 (=4) est inférieur à 7. Reste 3. 2^1 (=2) est inférieur à 3. Reste 1. 2^0 est égal à 1. Reste 0. Nous avons maintenant relevé toutes les puissances de 2 qui composent notre nombre. Il faut simplement écrire un 1 sous ces puissances de 2 et un 0 sous les autres :

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	1	0	0	1	1	1

Bien sûr, si vous faites à nouveau la somme des puissances de 2 sous lesquelles figure un 0, vous retrouverez le nombre décimal 167 dont nous sommes parti.

1.5.2 BIT ET OCTET

Nous avons pris plus haut un nombre décimal inférieur à 255. En effet, dans ce cas, 8 chiffres, ou 8 puissances de 2, suffisent pour représenter ce nombre en système binaire. La plus petite unité d'information que traite un ordinateur est appelée bit (BInary digiT). Un bit peut avoir deux états et peut donc représenter deux valeurs: 0 et 1. On parle de bit mis pour la valeur 1 et de bit non mis ou nul pour la valeur zéro. La réunion des huit bits forme un groupe appelé **OCTET**.

1.5.3 LE SYSTEME HEXADECIMAL

Le système hexadécimal repose sur la base 16. Il faut donc 16 chiffres pour représenter les nombres de ce système. On prend bien sûr les chiffres du système décimal pour représenter les valeurs de 0 à 9 mais on prend les 6 premières lettres de l'alphabet pour représenter les valeurs 10 à 15. La suite de nombres décimaux suivante :

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 etc...

devient en hexadécimal :

1 2 3 4 5 6 7 8 9 A B C D E F 10 11 12 13 etc...

Voici maintenant quelques exemples pour nous familiariser avec le maniement des nombres hexadécimaux. Convertissons tout d'abord les nombres hexadécimaux en nombre décimaux :

$$2\ E\ 0\ C = 2*16^3 + 14*16^2 + 0*16^1 + 12*16^0$$

$$= 2*4096 + 14*256 + 0*16 + 12*1 = \mathbf{11788}$$

Vous voyez que nous suivons toujours le même principe pour convertir un nombre d'un système numérique dans un autre. Un autre exemple :

$$0\ A\ B\ C = 0*16^3 + 10*16^2 + 11*16^1 + 12*16^0$$

$$= 0*4096 + 10*256 + 11*16 + 12*1 = \mathbf{2748}$$

L'intérêt principal du système hexadécimal vient du fait que la base de ce système, 16, est une puissance de 2 et se rapproche donc plus que le système décimal du système numérique binaire directement compris et utilisé par l'ordinateur. C'est pourquoi la conversion des nombres binaires ne pose plus de gros problèmes si on passe par le "détour" du système hexadécimal. Voici quelques exemples :

Binaire	**Hexadécimal**	**Décimal**
0101 1011 =	5B = $5*16^1 + 11*16^0$ =	91
1100 0011 =	C3 = $12*16^1 + 3*16^0$ =	195
1010 1010 =	AA = $10*16^1 + 10*16^0$ =	170

Vous avez certainement remarqué que nous avons divisé les nombres binaires en deux moitiés de 4 chiffres chacune. Chaque moitié a été ensuite convertie en un nombre hexadécimal. Dans le premier exemple, le premier et le troisième bits (en partant de la droite) de la moitié de gauche étaient mis, ce qui donne 5 en hexadécimal. Dans la moitié de droite, les premier, second et quatrième bits étaient mis, ce qui donne B en hexadécimal.

C'est ainsi que nous obtenons la valeur hexadécimale 5B.

Un groupe de quatre bits, qui peut donc être représenté soit par quatre chiffres binaires, soit par un chiffre hexadécimal, s'appelle un quartet.

A travers ces exemples vous voyez comment il faut procéder pour convertir des nombres d'un système numérique dans un autre.

Voici maintenant en conclusion comment convertir les nombres décimaux en nombres hexadécimaux. Le principe à suivre est le même que pour la conversion des nombres décimaux en nombres binaires.

Pour convertir par exemple le nombre décimal 49153 en son équivalent hexadécimal, il faut rechercher la plus grande puissance de 16 inférieure à 49153 : 16^4 (=65536) est supérieur à 49153 mais 16^3 (=4096) est encore inférieur à 49153. Divisez maintenant 49153 par 16^3, vous obtenez 12, il reste 1. Nous avons presque fini : 16^2 (=256) et 16^1 (=16) sont supérieurs au reste 1 mais 16^0 (=1) est égal à 1. Le reste 1, divisé par 160 donne 1, reste 0 :

$49153 = 12*16^3 + 0*16^2 + 0*16^1 + 1*16^0$

-le nombre décimal 12 correspond au chiffre hexadécimal C
-le nombre décimal 0 correspond au chiffre hexadécimal 0
-le nombre décimal 0 correspond au chiffre hexadécimal 0
-le nombre décimal 1 correspond au chiffre hexadécimal 1

Nous avons ainsi obtenu l'équivalent de 49153 en hexadécimal :

C001

1.6 LES OPERATEURS LOGIQUES

Vous rencontrerez les opérateurs logiques, également appelés opérateurs de BOOLE, d'après G. Boole (George Boole, mathématicien anglais né à Lincoln le 12.11.1815, mort à Cork (Irlande) le 8.12.1864 ; considéré comme le principal initiateur de la logique mathématique), presque dans chaque programme. Ce n'est que grâce à ces opérateurs logiques que les comparaisons et les manipulations de bits deviennent possibles. Le BASIC de l'Atari ST vous fournit les trois opérations logiques de base

NOT, AND et OR.

Ces trois opérateurs suffisent pour réaliser y compris les opérations logiques les plus complexes. C'est ainsi que la fonction **XOR** n'est qu'une combinaison de ces trois opérations, comme nous le verrons plus tard. En électronique, on retrouve ces opérateurs dans les circuits intégrés, dans les combinaisons les plus diverses (par exemple, portes logiques **NAND, NOR** et **EXOR**).

Comme nous le savons déjà, l'ordinateur ne peut distinguer que deux états, allumé et éteint. Cela a pour conséquence que l'ordinateur ne dispose également que d'une logique d'affirmation à deux valeurs. Il peut simplement décider si une affirmation est **vraie** ou **fausse**. Une affirmation serait par exemple :

2 < 3 (deux inférieur à trois)

Nous avons ici une affirmation VRAIE. L'ordinateur ne nous communique toutefois pas cette décision en sortant le message "vrai" ou "faux" mais il nous indique cette décision par des valeurs numériques appropriées. Si une affirmation est VRAIE, comme dans l'exemple ci-dessus, l'ordinateur nous envoie une valeur **DIFFERENTE DE 0**. Entrez la séquence d'instructions suivante dans l'ordinateur :

PRINT 2 < 3 (RETURN)

Sortie :

-1

La valeur obtenue est **DIFFERENTE DE ZERO**. L'ordinateur indique ainsi qu'il s'agit, pour lui, d'une affirmation **VRAIE**. Dans la plupart des cas, une affirmation vraie donnera comme résultat la valeur -1. Produisons maintenant une affirmation **FAUSSE** :

PRINT 3 < 2 (RETURN)

Sortie :

0

La valeur ZERO. L'ordinateur indique ainsi qu'il s'agit, pour lui, d'une affirmation **FAUSSE**. Une affirmation fausse donne toujours, comme résultat, **la VALEUR ZERO**.

Les trois opérateurs logiques font subir une opération à deux valeurs en examinant celles-ci bit par bit. Décrivons maintenant les différents opérateurs logiques.

1.6.1 NOT

L'opérateur **NOT** a pour effet de transformer une affirmation vraie en une affirmation fausse et vice versa. Le tableau suivant illustre cela :

Opérateur	Valeur 1	Valeur 2	Résultat
NOT	-1	-	0
	0	-	-1

Exemple :

PRINT NOT 0

Sortie :

-1

PRINT NOT -1

Sortie :

0

1.6.2 AND

L'opérateur **AND** ne donne pour résultat une affirmation vraie que si les deux conditions sont vraies.

Opérateur	Valeur 1	Valeur 2	Résultat
AND	0	0	0
	0	-1	0
	-1	0	0
	-1	-1	-1

Exemple :

PRINT 0 AND 0,0 AND 1,1 AND 0,1 AND 1

Sortie :

0 0 0 1

Un autre exemple illustrera la fonction AND.

Exemple :

PRINT 23 AND 12

Sortie :

4

Pour comprendre pourquoi nous obtenons ce résultat, examinons les modèles bits des valeurs 12 et 23.

Le modèle bits de 23 est

00010111.

Le modèle bits de 12 est

00001100.

Relions maintenant ces deux modèles au moyen de AND.

```
00010111

        > AND

00001100

= 00000100  =  4
```

1.6.3 OR

L'opérateur OR produit une affirmation vraie dès qu'une des deux affirmations est vraie.

Opérateur	Valeur 1	Valeur 2	Résultat
OR	0	0	0
	0	-1	-1
	-1	0	-1
	-1	-1	-1

Exemple :

PRINT 0 OR 0,0 OR 1,1 OR 0,1 OR 1

Sortie :

0 1 1 1

Un autre exemple illustrera la fonction OR.

Exemple :

PRINT 23 OR 12

Sortie :

31

Pour comprendre pourquoi nous obtenons ce résultat, examinons les modèles bits des valeurs 12 et 23.

Le modèle bits de 23 est

00010111.

Le modèle bits de 12 est

00001100.

Relions maintenant ces deux modèles au moyen de OR.

```
   00010111

                > OR

   00001100

= 00011111  =  31
```

De même que les opérateurs arithmétiques, les opérateurs logiques suivent des règles de priorité. L'ordre de priorité est le suivant: **NOT** a le premier rang, **AND** le second rang et **OR** le dernier rang de priorité. Cela signifie concrètement qu'une négation sera exécutée avant qu'une opération AND ou OR soit exécutée. Cet ordre de priorité peut bien sûr être modifié, comme pour les opérateurs arithmétiques, par la mise entre parenthèses de certaines expressions logiques.

En conclusion de ce chapitre sur les opérateurs logiques, nous allons vous présenter encore les fonctions **XOR** (eXclusive OR = ou exclusif), **EQV** (EQuiValence) et **IMP** (IMPlication).
De par leur mode d'opération, ces fonctions relèvent entièrement de ce chapitre.

1.6.4 XOR

Comme nous l'avons déjà indiqué, cette fonction se compose d'une combinaison des trois opérateurs. Mais examinons d'abord la fonction de **XOR**.

Dans le langage de tous les jours, c'est le plus souvent ce ou exclusif que nous utilisons. Par exemple, quand vous dites à un ami :

"Je viendrai en voiture ou en bus", ces deux possibilités s'excluent mutuellement car vous ne pouvez bien sûr pas venir à la fois avec votre voiture et en bus. Soit vous prendrez votre voiture et vous ne prendrez donc pas le bus, soit vous prendrez le bus et vous ne prendrez donc pas votre voiture. On n'obtient ainsi, avec la fonction XOR, une affirmation vraie que si les deux affirmations reliées par l'opération possèdent une qualité de vérité différente.

Vous pouvez vérifier que la fonction XOR travaille bien ainsi en entrant dans l'ordinateur la ligne suivante :

PRINT 0 XOR 0,0 XOR 1,1 XOR 0,1 XOR 1

Sortie :

0 1 1 0

La table de la fonction XOR se présentera donc ainsi :

Opérateur	Valeur 1	Valeur 2	Résultat
XOR	0	0	0
	0	-1	-1
	-1	0	-1
	-1	-1	0

Nous allons maintenant, comme nous vous l'avons promis, vous révéler l'opération de Bool qui permet de réaliser la fonction XOR. La fonction XOR se compose des trois opérations logiques de base, d'après la formule suivante :

Q = (X AND NOT Y) OR (NOT X AND Y)

Q est ici le résultat de l'opération, X et Y valent 0 ou 1.

1.6.5 EQV

L'opérateur logique **EQV** est pour ainsi dire la négation de l'opérateur XOR.
La table de la fonction EQV se présentera donc ainsi :

Opérateur	Valeur 1	Valeur 2	Résultat
XOR	0	0	-1
	0	-1	0
	-1	0	0
	-1	-1	-1

Donc, avec EQV, on obtiendra une affirmation vraie si et seulement si les deux affirmations à relier possèdent une même qualité de vérité.

1.6.6 IMP

L'opérateur logique **IMP** produit dans trois cas sur quatre une affirmation vraie et donc dans un cas seulement une affirmation fausse, de sorte que la table se présente ainsi pour cet opérateur :

Opérateur	Valeur 1	Valeur 2	Résultat
IMP	0	0	-1
	0	-1	-1
	-1	0	0
	-1	-1	-1

Il serait peut-être temps de mettre en pratique ces notions théoriques très arides. Essayez donc de résoudre les petits problèmes de la page suivante. Si vous hésitez sur la marche à suivre, consultez à nouveau le chapitre correspondant. Les solutions se trouvent à la fin du livre.

EXERCICES

1) Convertissez les nombres binaires suivants en nombres hexadécimaux:

   a) 01101100 b) 10010010
   c) 10111010 d) 11110000
   e) 00001100 f) 11001001

2) Convertissez les nombres hexadécimaux suivants en nombres décimaux :

   a) F0CA b) 1268
   c) 35A0 d) 0255
   e) F000 f) 0800

3) Convertissez les nombres binaires suivants en nombres décimaux :

   a) 10110111 b) 00110011
   c) 11111110 d) 00010101
   e) 01010101 f) 10101010

4) Convertissez les nombres décimaux suivants en nombres hexadécimaux :

   a) 63280 b) 24576
   c) 32769 d) 43981
   e) 65534 f) 18193

CHAPITRE 2

LA PROGRAMMATION EN BASIC

INTRODUCTION A LA PROGRAMMATION EN BASIC

Ce chapitre a pour but de vous apprendre l'utilisation des instructions BASIC au moyen de petits programmes d'exemple. Nous suivrons les cinq étapes du travail de programmation que nous avons décrites uniquement dans le premier programme. Nous nous concentrerons ensuite sur le troisième point de ce travail, la traduction de l'algorithme en BASIC.

2.1 LE PREMIER PROGRAMME BASIC

Supposons maintenant que monsieur Durand ne veuille plus calculer le volume d'une sphère mais la surface de cette sphère pour 10 rayons différents. Il va suivre cette fois nos indications à la lettre et il commence donc par faire l'analyse du problème :

1) Définition du problème

Il veut que son Atari ST, à partir des rayons qui lui seront fournis en centimètres, calcule la superficie S d'une sphère. La formule est :

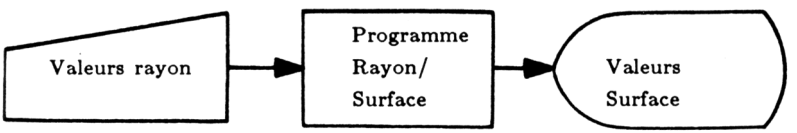
$$S=4PIr^2$$

2) Projet d'algorithme de solution

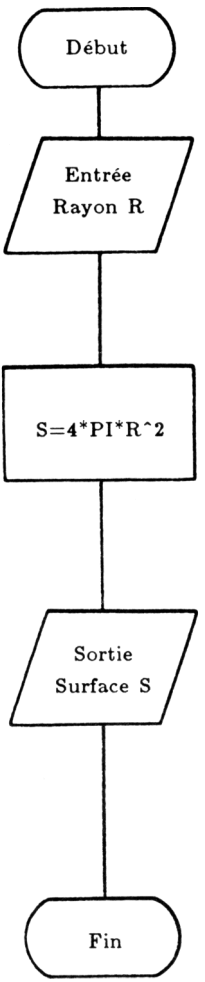
1. Début
2. Entrée de r
3. Calcul de S = $4PIr^2$
4. Sortie de S sur l'écran
5. Fin

Vous verrez à la page suivante le plan de flux de données et l'organigramme correspondant à cet algorithme. Cet organigramme est un organigramme linéaire car il ne prévoit pas de branchements sous forme de sous-programmes ou de boucles. Si vous ne comprenez pas pour le moment le sens des concepts de sous-programme et de boucle, ne vous inquiétez pas, nous les expliquerons au chapitre suivant.

PLAN DE FLUX DES DONNEES



ORGANIGRAMME



3) Réalisation du programme

```
10 INPUT"QUEL RAYON EN CM";R
20 LET S=4*3.14159*R^2
30 PRINT S
40 END
```

4) Test du programme

Il faudrait normalement vérifier d'abord notre plan de flux de données et notre organigramme. Ceci est cependant vite fait, vu la taille très réduite de notre programme. Vous pouvez alors lancer le programme avec l'instruction RUN.

5) Documentation

Le but de la documentation est de permettre à n'importe quel autre programmeur de pouvoir comprendre et donc modifier utilement le programme. Pour un programme aussi petit que le nôtre, le plan de flux de données et l'organigramme ainsi que la courte description du programme fournie au point 1 suffisent amplement.

Vous remarquez en examinant le listing de notre programme (point 3) que chaque instruction occupe une ligne de programme. Il vaut mieux en effet éviter d'écrire des lignes pleines de nombreuses instructions car vous risquez alors de ne plus pouvoir facilement comprendre votre programme et vous risquez également d'avoir du mal à le modifier si nécessaire.
Il est également très recommandé de numéroter les lignes de vos programmes de 10 en 10 comme nous l'avons fait de façon à faciliter l'ajout éventuel de lignes supplémentaires entre deux lignes existantes. Vous pouvez bien sûr programmer d'abord la ligne 20 puis ensuite seulement la ligne 10 puisque l'ordinateur classe automatiquement et exécute les lignes de programme dans l'ordre croissant des numéros de ligne.

Examinons maintenant les instructions employées dans notre programme: **INPUT, LET, PRINT** et **END.**

2.1.1 ENTREE DE VALEURS AVEC INPUT

L'instruction **INPUT** permet dans un programme l'entrée de valeurs pendant le déroulement du programme. L'utilisateur du programme (et non pas seulement le programmeur) peut donc exercer une influence directe sur le déroulement du programme. L'instruction INPUT peut être suivie d'un message placé entre guillemets comme c'est le cas dans notre programme d'exemple. Ce message doit être suivi d'une ou plusieurs variables. La première variable doit être séparée du message par un point-virgule et chaque variable doit être séparée de la suivante par une virgule. Lorsque le programme rencontre l'instruction INPUT, le curseur apparaît à l'écran et le programme est interrompu jusqu'à ce que vous entriez une entrée avec les touches du clavier, en marquant la fin de cette entrée en appuyant sur la touche RETURN. Voici quelques exemples pour plus de clarté :

a) **INPUT S** **Entrée : 1**

La valeur 1 sera ici affectée à la variable S.

b) **INPUT"QUEL RAYON";S** Entrée : 3

Le texte QUEL RAYON s'affiche à l'écran et après que vous ayez appuyé sur les touches 3 et ENTER, la valeur 3 sera affectée à la variable S.

c) **INPUT A,B,C** **Entrée : 4.3, .5, 4**

Les valeurs 4.3, 0.5 et 4 seront respectivement affectées au variables A, B et C. Souvenez-vous que les décimales sont placées après un point et non après une virgule comme dans la notation arithmétique française normale. N'oubliez pas que les virgules servent toujours à séparer entre elles différentes variables.

D'autre part, si vous placez une virgule entre le message suivant l'instruction INPUT et la première variable, le point d'interrogation placé automatiquement à la suite d'un message INPUT n'apparaîtra pas.

A RETENIR :

Avec INPUT les différentes variables sont séparées par des VIRGULES. Les décimales sont placées après un POINT.

L'instruction **LINE INPUT** est une variante de l'instruction INPUT. L'instruction LINE INPUT permet d'entrer une virgule comme partie de la variable entrée. Ceci entraîne qu'on ne peut affecter de valeur qu'à une seule variable avec LINE INPUT.
Le point d'interrogation n'est pas affiché automatiquement à l'écran.

A RETENIR :

L'instruction LINE INPUT ne permet d'affecter de valeur qu'à une seule variable. Lors de la sortie, aucun point d'interrogation n'apparaît (sauf si vous en placez un dans le message entre guillemets). Le texte à affecter à une variable alphanumérique (chaîne de caractéres) peut comprendre une virgule (voir plus loin).

2.1.2 AFFECTATION D'UNE VALEUR AVEC LET

L'instruction **LET** permet d'affecter une valeur à une variable. L'expression qui se trouve à la droite du signe égale est d'abord calculée par l'ordinateur puis la variable dont le nom figure à droite du signe égale reçoit la valeur de cette expression.

Le mot LET n'est cependant plus utilisé sur les ordinateurs comme l'Atari ST qui comprennent aussi bien le signe égale utilisé seul. Il faut cependant savoir que certains langages BASIC exigent l'emploi du mot LET.

Quelques exemples de cette instruction :

a) **LET A=10** ou A=10

affecte la valeur 10 à la variable A.

b) **LET A=A+5** ou A=A+5

affecte à la variable A l'ancienne valeur de A plus 5.

c) **LET A=A*B-8** ou A=A*B-8

L'ancienne valeur de la variable A est multipliée par la valeur de la variable B. On retranche 8 de ce produit et le résultat est la nouvelle valeur affectée à la variable A.
L'expression à la droite du signe égale peut comprendre autant de signes mathématiques et de variables ou constantes que vous voulez. On peut donc utiliser des formules mathématiques jusqu'à un certain degré de complexité (voyez notre programme pour calculer la superficie d'une sphère).
Il doit toujours y avoir une seule variable à la gauche du signe égale.

A RETENIR :

LET affecte une valeur à une variable. Il ne doit y avoir qu'une seule variable à la gauche du signe égale. A droite du signe égale peut figurer n'importe quelle expression mathématique.

Examinons à nouveau l'expression de l'exemple b) :

A=A+5 est absurde en mathématiques, il s'agit d'une équation fausse. Mais cette expression n'est pas une équation en BASIC mais une instruction, une affectation de valeur. Il faut considérer que les variables sont des sortes de tiroir. A=A+5 signifie qu'il faut ajouter 5 dans le tiroir A. Si ce tiroir A contenait déjà 4, il contiendra 9 une fois que l'instruction aura été exécutée. Vu sous cet angle, c'est beaucoup plus simple, non ?

2.1.3 SORTIE AVEC PRINT

L'instruction **PRINT** est certainement l'une des toutes premières instructions utilisées par les débutants programmeurs. C'est toutefois une des instructions les plus puissantes du BASIC de votre Atari ST. Elle vous permet d'afficher des textes, des messages ou les valeurs de diverses variables. Vous pouvez également mélanger les valeurs de variables avec des textes d'explication de la signification de ces variables.

Voici quelques exemples de l'utilisation de l'instruction PRINT. Les variables utilisées auront les valeurs suivantes :

A=10 : B=20 : C=30

EXEMPLES :

	Instruction	Sortie
a)	PRINT A	10
b)	PRINT "A"	A
c)	PRINT A*B	200
d)	PRINT A,B	10 20
e)	PRINT B;C	20 30
f)	PRINT "B";B	B 20
g)	PRINT "A VAUT";A	A VAUT 10

Nous arrêterons là les exemples d'utilisation de l'instruction PRINT pour le moment, mais nous y reviendrons souvent dans nos programmes d'exemple.

D'autre part l'Atari ST connait deux modes d'exécution des instructions : le mode direct et le mode programme. Le mode direct s'écrit comme les exemples que nous vous donnons ci-dessus. Si vous entrez donc PRINT A et que vous appuyez ensuite sur la touche RETURN, l'instruction PRINT A sera exécutée immédiatement.

Le mode programme se distingue du mode direct par le fait que chaque instruction est écrite sur une ligne commençant obligatoirement par un numéro de ligne. Si vous entrez par exemple :

10 PRINT A

et que vous appuyez ensuite sur la touche RETURN, cette ligne sera placée dans la mémoire BASIC de l'ordinateur. L'instruction PRINT A sera cette fois exécutée après le lancement du programme en mémoire en entrant RUN puis en appuyant sur la touche RETURN, et lorsque le programme arrivera à la ligne 10.

Venons-en maintenant à l'exemple a). Cette façon d'écrire permet de faire afficher à l'écran la valeur d'une variable. Un 10 apparaît sur l'écran puisque nous avions auparavant affecté 10 à la variable A.

Il faut noter en ce qui concerne la sortie des nombres avec PRINT que la valeur absolue d'un nombre est toujours précédée soit d'un espace s'il s'agit d'un nombre positif, soit d'un signe moins s'il s'agit d'un nombre négatif. De cette façon les nombres 10 et -10 ont toujours la même longueur lorsqu'ils sont affichés avec PRINT. Dans l'exemple a), comme aucun caractère ne suit le A, l'ordinateur effectue automatiquement, après avoir affiché la valeur de A, un retour de chariot (CARRIAGE RETURN) et un passage à la ligne (LINE FEED). Ceci signifie que la prochaine sortie avec PRINT commencera au début de la ligne suivante.

Dans l'exemple b), le A a été placé entre guillemets. Ceci entraîne que le A sera considéré comme du texte et non pas comme le nom d'une variable. C'est donc le caractère A qui sera affiché et non la valeur de la variable A. Tous les caractères placés entre guillemets sont affichés, sauf s'il s'agit de caractères de commande, par exemple pour le BIP.

L'exemple c) vous montre que vous pouvez également faire effectuer des calculs avec l'instruction PRINT. Le produit des variables A et B est ici calculé puis affiché sur l'écran. Suivent ici aussi un retour de chariot et un passage à la ligne.

L'exemple d) vous montre comment faire imprimer les valeurs de différentes variables sur une seule ligne. La virgule empêche donc ici le retour de chariot et le passage à la ligne. Elle influence cependant encore le format de sortie d'une autre façon.

Chaque ligne de sortie de l'Atari ST est divisée à nouveau en zones de 14 caractères. Ceci constitue donc ce qu'on appelle une tabulation. Si vous placez une virgule entre deux variables, la seconde variable sera donc placée au début de la zone de 14 caractères suivante donc en colonne 15. La troisième variable sera placée en colonne 29, et ainsi de suite.

Si vous entrez :

PRINT "1","2","3","4","5","6","7","8"

Si vous appuyez maintenant sur la touche RETURN, vous pourrez voir l'emplacement des diverses tabulations de l'écran. Si nous n'avions pas mis les chiffres entre guillemets, ils auraient été décalés d'une case sur la droite à cause du caractère réservé pour le signe des nombres.

L'exemple e) illustre l'emploi du point-virgule. Le point-virgule interdit le retour de chariot et le passage à la ligne mais également la tabulation.

Les expressions sont donc affichées à la suite les unes des autres, exactement comme elles ont été écrites, sans espace intercalaire. Ceci permet d'afficher un texte directement à la suite de la valeur d'une variable.

L'exemple f) vous montre comment, toujours grâce au point-virgule, vous pouvez faire afficher la valeur d'une variable directement à la suite d'un texte (ici le nom de la variable).

L'exemple g) utilise la même possibilité avec un texte explicatif. Cela permet de commenter plus précisément la sortie des résultats pour indiquer à l'utilisateur de quelle valeur il s'agit.

L'instruction END dans la dernière ligne de notre programme marque la fin logique du programme. Cette instruction n'est pas toujours placée dans la dernière ligne du programme car elle peut être suivie de sous-programmes. Mais nous n'en dirons pas plus pour le moment.

L'instruction END n'est d'ailleurs pas utile lorsque la fin logique du programme coïncide avec la dernière ligne du programme. L'ordinateur est en effet parfaitement capable de reconnaître la fin du programme. Il vaut mieux cependant placer cette instruction systématiquement à la fin de votre programme car c'est plus élégant et cela peut en outre vous éviter de légers problèmes si vous ajoutez plus tard des lignes de sous-programmes à votre programme.

Mais revenons pour l'instant à l'instruction PRINT, ou plutôt à une variante de cette instruction, l'instruction **PRINT USING**.

2.1.3.1 PRINT USING

Cette variante de l'instruction PRINT permet de réaliser des éditions "formatées" des valeurs ou textes que vous voulez sortir. Cette instruction est utilisée en liaison avec les caractères de commande suivants:

Pour les éditions numériques :

#	définit le nombre de chiffres à sortir
.	indique la position du point décimal
+	le signe + doit être sorti avec les nombres positifs
-	le signe - doit être sorti avec des nombres négatifs
	les caractères plus ou moins ne peuvent être utilisés que séparément.
**	remplir avec des étoiles à la place des espaces
$$	le premier caractère sorti doit être le signe $
**$	utilisation combinée de $ et *
,	une virgule doit séparer par groupes de trois les chiffres placés à la gauche du point décimal
^^^^	les valeurs doivent être sorties en écriture scientifique

Pour les éditions de texte :

!	seul le premier caractère du texte ou de la variable alphanumérique doit être sorti
\ \	le nombre de caractères sortis correspondra au nombre d'espaces entre les traits plus 2
&	le texte sera sorti en entier

Illustrons l'utilisation de l'instruction PRINT USING par quelques exemples. Entrez d'abord dans votre ordinateur :

```
a = 12345.678          <RETURN>
b = 34.3455            <RETURN>
c = -520               <RETURN>
d$= "Atari ST"         <RETURN>
```

Entrez maintenant les instructions PRINT USING suivantes en n'oubliant pas d'appuyer chaque fois sur la touche RETURN pour faire exécuter une instruction :

PRINT USING "#####.##";a

Vous obtenez en sortie sur l'écran la valeur de a arrondie à deux décimales :

12345.68

PRINT USING "#####.##";b

Vous obtenez en sortie sur l'écran la valeur de b arrondie à deux décimales :

34.35

Notez que les points décimaux de ces deux valeurs sont parfaitement alignés, ce qui n'aurait pas été le cas si vous aviez utilisé uniquement l'instruction PRINT.

Si le nombre à sortir sur l'écran dépasse le format indiqué dans l'instruction PRINT USING, un signe de pourcentage est affiché devant cette valeur pour indiquer que le format prescrit ne peut pas être respecté :

PRINT USING "####.##";a

Vous obtenez en sortie sur l'écran :

%12345.68

Si vous voulez souligner particulièrement les résultats négatifs ou positifs, vous pouvez utiliser les combinaisons suivantes :

PRINT USING "+######.##";a

sortie :

+12345.68

PRINT USING "#####.##+";a

sortie :

12345.68+

PRINT USING "#####.##-";c

sortie :

520.00-

Le signe moins est ici placé à la suite de la valeur absolue du nombre alors qu'il est normalement placé devant elle.

Voici maintenant comment vous pouvez remplir les espaces non utilisés de votre format d'étoiles. Il s'agit d'un format que vous connaissez peut-être déjà pour l'avoir rencontré sur des relevés de compte :

PRINT USING "######.##";a**

sortie :

****12345.68**

PRINT USING "#####.##";b**

sortie :

*****34.35**

Le format suivant vous permet d'indiquer des sommes en dollars :

PRINT USING "$$#####.##";a

sortie :

$12345.68

Vous pouvez bien sûr combiner le signe $ avec l'étoile :

PRINT USING "$#####.##";b**

sortie :

*****$34.35**

Vous pouvez d'autre part séparer les groupes de trois chiffres par une virgule et ajouter également des symboles supplémentaires tels que F :

PRINT USING "##,###.## F";a**

sortie :

***12,345.68 F**

Vous pouvez également faire éditer les nombres suivant l'écriture scientifique :

```
PRINT USING "##.##^^^^";a
```

sortie :

```
1.23E+04
```

Le signe '_' vous permet de sortir également les caractères de commande comme '#'.

```
PRINT USING "NUMERO DE FICHIER _###";b
```

sortie :

```
NUMERO DE FICHIER #34
```

Voyons maintenant les différents formats possibles pour la sortie de textes :

```
PRINT USING "!";d$
```

sortie :

```
A
```

Le point d'exclamation a donc pour effet d'imprimer uniquement le premier caractère de la variable alphanumérique (variable de texte) d$ sera sorti. Rappelez-vous que nous avons affecté le texte "Atari ST" à cette variable.

Le format suivant nous permet de définir le nombre de caractères d'une chaîne de caractères qui doivent être imprimés :

PRINT USING "\ \";d$

sortie :

Atari

Les guillemets entouraient 5 caractères (en comptant les traits). C'est pourquoi seuls les cinq premiers caractères de la variable d$ ont été imprimés. Cet emploi de PRINT USING est proche de l'instruction LEFT$ dont nous parlerons plus loin. Par ailleurs, pour obtenir ce caractère de commande, ce trait diagonal particulier, vous devez utiliser la combinaison des touches **ALTERNATE et \.**

Voici enfin comment faire imprimer l'intégralité d'un texte :

PRINT USING "&";d$

sortie :

Atari ST

Notez que vous pouvez également arriver au même résultat si vous utilisez une instruction PRINT normale :

PRINT d$

Il peut cependant parfaitement se produire que vous ayez besoin d'employer cette possibilité à l'intérieur d'une instruction PRINT USING.

Pour être complet, nous évoquerons encore l'instruction **WRITE**

Il s'agit d'une autre variante de l'instruction PRINT. Elle se distingue par le fait qu'elle sort les guillemets avec le texte :

WRITE "Atari ST"

donnera à l'écran :

"Atari ST"

Si vous utilisez l'instruction WRITE avec des variables ou des constantes séparées par des virgules, la virgule n'aura pas de fonction de tabulation et sera sortie avec les variables et constantes :

WRITE 2,3,4

donnera à l'écran :

2,3,4

Nous avons maintenant décrit toutes les instructions qui constituaient notre petit programme ainsi que deux instructions "cousines" de ces instructions. Nous espérons que vous n'aurez plus de difficulté à utiliser ces instructions.

Voyons maintenant comment l'instruction **REM** peut nous permettre de rendre nos programmes plus compréhensibles pour d'autres mais aussi, et peut-être surtout, pour nous-même.

2.1.4 LES COMMENTAIRES AVEC L'INSTRUCTION REM

REM vous permet de placer des commentaires (REMarques) en n'importe quel endroit de votre programme. Tout ce qui suit une instruction REM est ignoré par l'ordinateur lors de l'exécution du programme, même s'il s'agit d'instructions BASIC. L'instruction REM est donc un outil très puissant de documentation des programmes.

Nous allons donc compléter maintenant notre programme d'exemple, de façon à le rendre parfaitement compréhensible même pour ceux qui n'ont pas participé à son écriture :

```
10 REM CALCUL DE LA SURFACE D'UNE SPHERE
20 REM ENTREE DU RAYON EN CM
30 INPUT"QUEL RAYON EN CM";RAYON
40 REM CALCUL DE LA SURFACE
50 LET S=4*PI*RAYON^2
60 REM SORTIE DE LA SURFACE EN CM CARRES
70 PRINT"LA SURFACE DE LA SPHERE EST DE ";S;" CM2"
80 END
```

Les lignes 10 et 20 indiquent le but du programme et comment les entrées doivent être faites. En ligne 30 les données sont entrées avec INPUT. Un message est envoyé à l'utilisateur du programme. La ligne 40 indique dans son commentaire le but de la ligne 50 qui est de calculer la surface à partir de la donnée entrée. La ligne 50 affecte à la variable S la surface calculée à partir de la formule mathématique placée à la droite du signe égale. La ligne 60 indique que la ligne suivante affiche la surface en cm_2. La ligne 70 affiche la valeur de la variable S à la suite d'un texte explicatif. La ligne 80 termine le programme avec l'instruction END.

2.2 LES VARIABLES ET LEUR UTILISATION DANS LES PROGRAMMES

Avant que nous ne vous donnions quelques problèmes à résoudre, il nous faut encore décrire les différents types de variables du BASIC de votre Atari ST.

Votre ordinateur connait en effet trois types de variables. Le premier type de variable est celui des variables entières (integer variable).

Ce type de variable qui ne peut donc recevoir que des valeurs entières est désigné par le signe de pourcentage % placé à la suite du nom de la variable (par exemple A% ou C4%). Si vous tentez d'affecter à une variable entière un nombre comportant des décimales, seule la partie entière de ce nombre sera prise en compte. Il faut cependant tenir compte d'une contrainte supplémentaire pour utiliser ce type de variable : non seulement les valeurs affectées doivent être entières mais elles doivent en outre être comprises entre -32768 et 32767.

Le second type de variable, variables réelles (real variable) permet de représenter tous les nombres. Les noms des variables de ce type n'ont pas à être désignées par un signe particulier. On peut toutefois utiliser à titre facultatif le signe d'exclamation ! à cet effet. Lorsque vous venez d'allumer votre ordinateur, toutes les variables numériques sont d'abord considérées comme des variables entiéres.

Le troisième type de variable est désigné par le signe dollar $. Il s'agit des variables alphanumériques ou chaînes de caractères. Ces variables peuvent recevoir n'importe quel texte mais elles ne peuvent contenir plus de 255 caractères. Sinon, vous obtenez le message d'erreur :

Strings cannot be over 255 characters long

Dans le choix des noms des variables il est important de tenir compte du fait que le BASIC de l'Atari ST ne reconnait une variable que d'après les 31 premiers caractères de son nom. Vous pouvez donc parfaitement appeler une variable

'SOCIETENATIONALEDESCHEMINSDEFERFRANCAIS'

mais seuls les 31 premiers caractères seront significatifs. Beaucoup de micro-ordinateurs n'acceptent pas de noms aussi longs. Cette caractéristique très importante de l'Atari ST vous permet de donner à vos variables des noms caractéristiques qui vous

permettent de comprendre plus facilement leur rôle dans l'architecture générale de votre programme. C'est ainsi que nous avons dans notre exemple appelé RAYON la variable contenant le rayon de la sphére. Vous ne pouvez pas utiliser les mots clés BASIC (instructions, variables du systéme, etc...) comme noms de variables mais vous pouvez les utiliser dans les noms de vos variables : **AND** ne peut être une variable mais **WAND** peut parfaitement être le nom d'une variable.

Les chiffres peuvent également faire partie du nom des variables mais seulement à partir de la deuxiéme position : **1A** ou **2emecompte** ne peuvent pas être utilisés mais **A1** ou **compte9** sont parfaitement possibles comme noms de variables.

Vous devez également veiller à ne pas utiliser comme noms de variable des instructions comme :

OR, FN ou **ABS**

L'Atari ST utilise en outre les noms de variable suivants pour des fonctions internes :

GB, AS, ALL

Ces noms ne doivent pas non plus être utilisés dans vos programmes. Voilà ce que nous avions à dire sur les contraintes à respecter dans le choix des noms de variable.

2.2.1 OPERATIONS DE CALCUL AVEC DES VARIABLES

Si vous voulez effectuer des calculs avec vos variables, il faut tout d'abord connaître les rangs de priorité des différentes opérations de calcul :

Signe	Rang de priorité	Signification
^	premier rang	élévation à la puissance
*	second rang	multiplication
/	second rang	division
+	troisiéme rang	addition
-	troisiéme rang	soustraction

Les opérateurs logiques obéissent également à des régles de priorité que nous avons déjà expliquées.

Vous en savez maintenant suffisamment pour résoudre les exercices qui vous sont proposés dans les pages suivantes. Ces exercices comportent d'une part des questions sur les chapitres précédents et d'autre part des petits programmes à réaliser. Essayez d'abord de résoudre ces problèmes sans vous reporter aux chapitres précédents. N'ayez surtout pas peur de faire des erreurs car la recherche des erreurs constitue de toute façon une partie importante de tout travail de programmation. La seule chose qui importe est que vous essayiez vraiment de faire ces exercices par vous-même afin que vous cherchiez à comprendre ensuite quelles erreurs vous avez faites éventuellement et pourquoi vous les avez faites. Nous vous invitons d'autre part à bien suivre les 5 étapes du travail de programmation que nous avons décrites. Avant d'entrer un nouveau programme, n'oubliez pas d'entrer l'instruction NEW qui supprime le programme en mémoire. Nous vous souhaitons beaucoup de succès dans votre travail sur ces exercices.

EXERCICES

1. Indiquez si les noms de variables suivants sont autorisés et justifiez votre point de vue :

a) X1	b) Date$	c) AUTO
d) OU%	e) IF	f) GB
g) 4Nombre%	h) 255	i) LUNDI

2. Ecrivez un programme qui entre quatre valeurs A, B, C et D et qui sort les valeurs A et B l'une derrière l'autre sur une seule ligne et les valeurs C et D dans la ligne suivante avec une tabulation.

3. Ecrivez un programme qui calcule la surface d'un triangle rectangle en mètres carrés et sort le résultat avec un message d'explication.

4. Ecrivez un programme qui fournisse le poids idéal d'un être humain (taille en cm moins 100 moins 10 pour cent). La taille doit être entrée en centimètres et le poids doit être affiché en kilogrammes.

5. Ecrivez un programme calculant le nombre de litres dans un aquarium aprés que le programme ait demandé qu'on lui indique les données en cm correspondant à la longueur, la hauteur et la largeur.

6. Modifiez le programme de l'exercice 2 de façon à ce que chaque valeur soit écrite dans une ligne distincte avec un texte d'accompagnement.

2.3 LES FONCTIONS NUMERIQUES

Nous allons aborder dans ce chapitre les fonctions mathématiques intégrées que possède votre ordinateur. Il nous faut pour cela faire un petit détour à travers le domaine des mathématiques. Ne vous inquiétez pas il ne s'agit pas pour nous de faire un cours de mathématiques, mais uniquement de vous permettre de comprendre à quoi servent ces fonctions mathématiques intégrées.

Dans de nombreux ouvrages sur le BASIC ainsi que dans le manuel de votre Atari ST, il est indiqué que les valeurs des fonctions trigonométriques SIN(X), COS(X) ou TAN(X) sont fournies conformément à la norme, en radian. Qu'est ce que cela signifie?

Il s'agit de l'indication de l'angle avec lequel le SINUS ou le COSINUS doit être calculé. Comme vous le savez certainement, un cercle est divisé en 360 degrés. 1 degré est donc le 360ème d'un cercle. 90 degrés correspondent donc à un quart de cercle, 180 degrés à un demi-cercle, etc...

Pour calculer le radian, on ne divise pas le cercle en 360 degrés, mais on part de l'unité que représente le cercle, pour calculer la proportion du cercle représentée par un angle, d'après la formule :

U=2*PI*R

La formule est simplifiée en prenant un cercle unitaire (rayon=1) :

U=2*PI*1 ou **U=2*PI**

Les angles sont donc ainsi mesurés non pas en degrés mais en "RAD" (radian). Dans notre exemple, le cercle possèderait 360 degrés ou 2*PI (6.2831...) RAD. 90 degrés correspondraient à 2*PI/4 ou PI/2 RAD. L'intérêt de cette mesure des angles est qu'elle permet, à partir de la valeur obtenue pour un rayon de 1, de déterminer directement la longueur de l'arc de cercle.

Ce type de calcul nécessite une certaine adaptation au départ car il est plus facile de se représenter un quart de cercle sous la forme de 90 degrés que de PI/2 RAD.

Cette brève incursion dans les mathématiques suffira pour l'instant. Voici maintenant quelques petits programmes d'exemple qui vous rendront ces concepts peut-être un peu plus clairs.

Entrez ce programme dans votre ordinateur :

```
10 INPUT"ENTREE EN DEGRES";DEGRES
20 REM CALCUL DU SINUS
25 PI=3.1415927
30 SI=SIN(DEGRES*PI/180)
40 PRINT"LE SINUS DE";DEGRES;"DEGRES ";
50 PRINT"EST =";SI
60 END
```

Lancez maintenant ce programme avec RUN et entrez pour l'angle la valeur 90. Vous devez obtenir 1 comme résultat. Le programme attend que vous entriez un angle exprimé en degrés et il calcule le sinus correspondant. Comme, sur l'Atari ST, la variable PI n'est pas disponible directement, celle-ci doit d'abord être définie, en ligne 25. Si vous voulez donc pouvoir entrer dans vos programmes des angles en degrés, vous devez utiliser la conversion réalisée en ligne 30.

Pour calculer le cosinus, il suffirait de remplacer en ligne 30 SIN par COS. Vous pouvez d'ailleurs conserver le nom de la variable SI. Pour voir maintenant la différence avec les mesures en radian, entrez le programme suivant, non sans avoir au préalable fait exécuter l'instruction NEW :

```
10 INPUT"ENTREE EN RADIANS";RADIAN
20 REM CALCUL DU SINUS
30 SI=SIN(RADIAN)
40 PRINT"LE SINUS DE";RADIAN;"RADIANS ";
50 PRINT"EST =";SI
60 END
```

Comme vous le voyez, par rapport à notre programme précédent, la ligne 25 a disparu; la ligne 30 a été modifiée. Lancez le programme et entrez la valeur 1.57079633 (PI/2 = quart de cercle). Vous obtenez la valeur 0.

L'utilisation des autres fonctions est aussi simple. Il suffit de fournir à ces fonctions la valeur sur laquelle elles doivent effectuer un calcul. SQR(X) calcule la racine carrée de X, ATN(X) la tangente de X, EXP(X) la Xième puissance de e=2.71828183 et LOG(X) le logarithme de X de base e. Ces deux dernières fonctions sont d'effet exactement inverse :

si vous entrez :

PRINT EXP(1)

vous obtenez 2.71828.

Si vous entrez maintenant :

PRINT LOG(2.71828183)

vous obtenez à nouveau 1. Si vous voulez faire calculer le logarithme de base 10, il vous suffit de faire remplacer LOG(X) par LOG10(X). Le logarithme de base e est aussi appelé logarithme néperien. Le programme d'exemple suivant calcule pour vous aussi bien le logarithme néperien que le logarithme décimal :

```
10 INPUT"ENTREE DU NOMBRE";nombre
20 REM CALCUL DU LOGARITHME NEPERIEN
30 LN=LOG(nombre)
40 REM CALCUL DU LOGARITHME DECIMAL
50 LO=LOG10(nombre)
60 PRINT"LE LOGARITHME NEPERIEN ";
70 PRINT"DE";nombre;" EST DE";LN
80 PRINT
90 PRINT"LE LOGARITHME DECIMAL ";
100 PRINT"DE";nombre;" EST DE";LO
110 END
```

Vous voyez que le maniement de ces fonctions dans des programmes (pour peu bien sûr qu'on sache à quoi elles correspondent en mathématiques) est très simple. La seule petite difficulté réside dans le traitement différent des degrés et des radians.

A RETENIR :

Les fonctions trigonométriques attendent des valeurs en radians. Pour effectuer un calcul en degrés, il faut convertir ces valeurs par un calcul approprié.

Les fonctions LOG et EXP se rapportent respectivement au logarithme de base e et à l'exponentielle.

La fonction SGN(X) fournit le signe de X. Le résultat est 1 pour un nombre positif, 0 pour la valeur 0 et -1 pour un nombre négatif.

La fonction INT(X) fournit la partie arrondie de X. Avec une petite routine utilisant cette instruction, il est donc facile d'arrondir des nombres au nombre de décimales que vous voulez. C'est ce que réalise le petit programme suivant :

```
10 INPUT"COMBIEN DE CHIFFRES APRES LA VIRGULE";X%
20 INPUT"QUEL NOMBRE";nombre
30 REM ARRONDIR
40 nombre=INT(nombre*10^X%+.5)/10^X%
50 REM SORTIE DU NOMBRE ARRONDI
60 PRINT nombre
70 END
```

La ligne 10 vous demande à combien de décimales devra être arrondi le nombre. Le nombre de décimales est affecté à la variable X%. Nous utilisons une variable entière puisqu'il ne peut pas y avoir de nombre non entier de décimales. La ligne 20 vous demande alors d'entrer un nombre quelconque. Entrez ici n'importe quel nombre décimal dont le nombre de décimales soit supérieur au nombre de décimales auquel vous voulez l'arrondir.
La ligne 40 arrondit ce nombre. Ce nombre est d'abord multiplié par 10 puissance X%, ce qui place les décimales que vous voulez conserver à la gauche du point décimal. .5 (1/2) est alors ajouté au nombre à arrondir car la fonction INT se contente de supprimer les décimales en trop, c'est-à-dire d'arrondir systématiquement à l'unité inférieure. En ajoutant .5, vous êtes sûr que votre nombre sera arrondi à l'unité la plus proche. Le nombre est enfin divisé par 10 puissance X% ce qui donne comme résultat final votre nombre arrondi au nombre de décimales que vous avez indiqué.

Lancez le programme avec RUN (RETURN). Entrez quelques valeurs pour voir à quels résultats vous arrivez. Essayez surtout de bien comprendre la logique de la ligne 40 qui réalise le calcul de la valeur arrondie. Vous pourrez ainsi utiliser cet algorithme dans vos programmes lorsque vous aurez besoin d'arrondir des nombres.
Evoquons encore brièvement une autre instruction pour arrondir des nombres :

CINT

Cette instruction arrondit les nombres ou les variables comme INT

mais elle convertit en outre la valeur arrondie en format entier. Les nombres ainsi convertis doivent donc être entre -32768 et 32767.

PRINT CINT(-35.6)

Sortie :

-36

Les fonctions CSNG et CDBL convertissent des variables en double précision (4 octets) et des variables en précision simple (2 octets) et vice versa.

La fonction **MOD** fournit le reste d'une division :

PRINT 32 MOD 7

donnera le reste de 32 divisé par 7, soit 4.

2.3.1 FONCTIONS AVEC DEF FN

L'instruction **DEF FN** vous fournit un moyen pratique d'économiser de la place en mémoire. Cette instruction permet d'affecter des fonctions mathématiques complexes à l'expression FN. Cette expression peut en effet être appelée ensuite pour calculer la valeur correspondant au paramètre qui lui est fourni, d'après la fonction qui a été définie avec DEF FN. Voici un exemple :

```
10 REM DEFINITION DE LA FONCTION
20 DEF FNF(X)=X^2 + 2*X + 4
30 REM ENTREE DU PARAMETRE
40 INPUT"QUELLE VALEUR";X
50 REM SORTIE
60 PRINT FNF(X)
70 END
```

La ligne 20 affecte la fonction mathématique X_{2+2X+4} à l'expression FNF(X). X représente le paramètre en fonction duquel le résultat FNF(X) sera calculé. Vous n'avez donc à écrire qu'une fois toute fonction, aussi complexe soit-elle, dont vous aurez à vous servir souvent dans votre programme. Il vous suffit ensuite d'utiliser le nom de la fonction.

Cette fonction vous permet donc de réaliser très facilement toutes sortes de tables mathématiques de calcul.

2.3.2 NOMBRES ALEATOIRES

Le BASIC de l'Atari ST possède un générateur aléatoire intégré que vous pouvez appeler avec l'instruction RND(X). Cette fonction peut vous permettre d'effectuer une simulation où le hasard joue un rôle ou bien sûr aussi de provoquer des évènements imprévisibles dans un programme de jeu. L'utilisation de cette fonction est très simple : A=RND(1) affecte à A une valeur aléatoire entre 0.0 et 1.0. Si vous utilisez comme valeur de X (entre parenthèses) des valeurs négatives, c'est toujours la même suite de nombres qui sera sortie. Le programme suivant simule un jet de dé. Chaque fois que vous lancez le programme, vous obtenez un nombre entre 1 et 6 :

```
10 REM PRODUIRE UN NOMBRE ALEATOIRE
20 A=INT(6*RND(1))+1
30 PRINT A
40 END
```

La ligne 20 utilise la fonction INT pour qu'il n'y ait pas de nombres avec décimales. Le nombre aléatoire est multiplié par 6 ce qui permet d'obtenir une valeur entre 0.0 * 6 = 0 et 1.0 * 6 = 6. Il s'agit là de bornes qui ne sont jamais atteintes. On a encore ajouté 1 pour avoir non pas des nombres compris entre 0 et 5 inclus mais entre 1 et 6 inclus.
Si vous vouliez maintenant obtenir des nombres aléatoires entre 100 et 150, il faudrait modifier ainsi la ligne 20 :

```
20 A=INT(50*RND(1))+100
```

Voici la formule générale pour obtenir un nombre compris entre "plancher" et "plafond" :

```
A=INT((plafond+1-plancher)*RND(1))+plancher
```

Dans les exemples plus simples, on ne reconnait pas toujours cette formule générale au premier coup d'oeil. La ligne suivante

```
20 A=INT(6*RND(1))+1
```

est en fait une simplification de la formule :

```
20 A=INT((6+1-1)*RND(1))+1
```

Vous voyez que, si la limite inférieure est 1, l'expression dans la première partie de la formule peut être simplifiée.

Si vous voulez que le générateur de hasard soit initialisé avec une nouvelle valeur de départ, vous pouvez utiliser l'instruction **RANDOMIZE** :

```
RANDOMIZE 45
```

affecte au générateur de hasard la nouvelle valeur de départ 45. Vous pouvez maintenant affecter à la variable A une nouvelle valeur aléatoire avec l'instruction :

```
A=RND(4)
```

2.3.3 AUTRES INSTRUCTIONS POUR MANIER LES VARIABLES

Le riche BASIC de l'Atari ST possède toute une série d'instructions permettant de définir, de modifier ou de supprimer des variables ou des formats de variables. Voici une brève description de ces instructions :
Souvenez-vous un instant du chapitre sur les systèmes numériques. L'Atari ST offre deux fonctions permettant de convertir les nombres décimaux en nombres binaires ou en nombres hexadécimaux.

La fonction

HEX$(X)

convertit un nombre décimal en un nombre hexadécimal. X est le nombre à convertir. Ce nombre doit être compris entre -32768 et +32767 :

PRINT HEX$(60)

donnera :

3C

La fonction

OCT$(X)

convertit un nombre décimal en un nombre octal (base 8). X est le nombre à convertir. Ce nombre doit être compris entre -32768 et +32767 :

PRINT OCT$(16)

donnera : **20**

Comme nous l'avons déjà indiqué, le BASIC de l'Atari ST connait trois différents types de variables (entières, réelles et alphanumériques). Si vous voulez définir à quel type de variables doivent appartenir certaines variables d'un programme, vous pouvez utiliser les instructions suivantes :

DEFDBL

DEFINT

DEFSNG

DEFSTR

Si vous entrez par exemple :

DEFSTR A-B

toutes les variables commençant par A ou B seront maintenant définies comme étant des variables alphnumériques. Vous n'aurez donc plus besoin de placer à la suite du nom de ces variables le signe dollar ($). L'instruction

DEFSNG C-D

définit toutes les variables commençant par C ou D comme des variables réelles. Les autres instructions fonctionnent de façon analogue.

Si vous ne voulez sortir que la partie d'une valeur située à la gauche du point décimal, vous pouvez utiliser l'instruction :

FIX

La différence entre FIX et INT est que FIX supprime systématiquement les décimales alors que INT arrondit les nombres

négatifs au premier nombre entier négatif plus petit :

PRINT INT(-3.55)

donnera :

-4

alors que

PRINT FIX(-3.55)

donnera :

-3

2.3.4 ASC(X$) ET CHR$(X)

L'Atari ST vous permet d'afficher à l'écran, avec l'instruction PRINT, des nombres, des lettres et certains caractères spéciaux, si ces derniers sont placés entre guillemets. Tous les caractères ne peuvent être toutefois affichés avec cette méthode. C'est la fonction CHR$ qui vous permet de résoudre de problème. Cette fonction vous permet en effet de faire sortir tous les caractères qui composent le jeu de caractères de votre ordinateur. Entrez en mode direct :

PRINT CHR$(65)

Si vous appuyez maintenant sur la touche RETURN, le caractère A apparaît maintenant à l'écran.
La fonction ASC est l'inverse de la fonction CHR$. Vous pouvez à tout moment indiquer à votre ordinateur de vous donner le code ASCII d'un caractère quelconque :

PRINT ASC("A")

donnera : 65 qui est le code ASCII de la lettre A. Le manuel de l'Atari ST vous indique à quels caractères correspondent les différents code ASCII.

Voici maintenant quelques exercices pour vous permettre d'utiliser votre nouvelle science. Ces exercices vous permettront en effet d'employer les instructions qui viennent d'être expliquées. Nous vous recommandons à nouveau de bien respecter les cinq étapes du travail de programmation et nous vous souhaitons beaucoup de réussite.

EXERCICES

1. Ecrivez un programme qui simule le jet de dés avec deux dés. Le résultat des deux dés doit être affiché sur une seule ligne avec une tabulation. N'oubliez pas de faire vider l'écran en début de programme.

2. Ecrivez un programme qui calcule la surface d'un triangle quelconque d'après la formule :
F=SQR(S(S-A)(S-B)(S-C)) où S=1/2(A+B+C). Réfléchissez au fait que vous ne pouvez pas entrer cette formule dans votre programme sous cette forme. Le programme doit demander l'entrée des valeurs A, B et C. Le résultat doit être présenté par un message d'explication.

3. Ecrivez un programme qui vous demande d'entrer un caractère et qui sorte ensuite ce même caractère suivi de son code ASCII.

4. Ecrivez un programme qui calcule une hauteur d'après le temps de chute d'un corps. Le programme doit demander l'entrée du temps de chute mesuré. La résistance de l'air (T) ne sera pas prise en compte. La formule est S=1/2GT^2. La constante G vaut 9.81. Le résultat sera sorti en mètres.

5. Ecrivez un programme qui calcule la consommation d'essence aux 100 kilomètres d'après la formule :

Consommation aux 100 kilomètres = consommation totale / nombre de kilomètres parcourus * 100

2.4 TAB et SPC

Ces deux instructions permettent d'afficher des données ou des caractères dans des emplacements déterminés de l'écran. L'instruction TAB et son paramètre entre parenthèses affichent le caractère de façon relative par rapport au début de la ligne de l'écran où se trouve actuellement le curseur :

PRINT TAB(15) "TEST"

placera le mot TEST à partir du 15ème caractère de la ligne de l'écran. Utilisez maintenant l'instruction SPC de la même façon, à la place de l'instruction TAB. Vous obtenez presque le même résultat. Prenons donc un exemple qui montre mieux la différence entre ces deux instructions. Entrez :

PRINT TAB(5) "TEST 1" TAB(20)"TEST 2"

Vous voyez que le mot TEST 1 se trouve à partir de la cinquième position de la ligne, et le mot TEST 2 à partir de la vingtième. Remplacez maintenant la deuxième TAB par SPC :

PRINT TAB(5) "TEST 1" SPC(20)"TEST 2"

Vous voyez tout de suite la différence: le mot TEST 2 n'a pas été affiché à partir de la vingtième position de la ligne mais à partir de la vingtième position après le dernier caractère de TEST 1. TAB se réfère donc à une position absolue sur la ligne de l'écran, et SPC à une position relative par rapport au dernier caractère sorti. Les valeurs transmises à ces fonctions ne doivent pas excéder 255.

Si vous utilisez ces instructions pour une sortie sur imprimante, notez que l'instruction TAB n'a pas d'intérêt dans ce cas car, en liaison avec l'instruction PRINT# elle sera ignorée ou interprétée comme l'instruction SPC. TAB ne doit donc être utilisé qu'avec des PRINT "normaux".

A RETENIR

TAB(X) compte le nombre d'emplacements à partir du premier emplacement de la ligne de l'écran actuelle.

SPC(X) ajoute en fait X espaces avant que ne reprenne l'affichage.

2.5 LES CHAINES DE CARACTERES

Une chaîne de caractères peut comporter 255 caractères sur l'Atari ST. Les variables contenant des chaînes de caractères, appelées aussi variables alphanumériques, sont marquées par le signe dollar ($). Pour affecter une chaîne de caractères à une variable alphanumérique, vous pouvez procéder exactement comme pour les variables numériques, avec cette seule différence que le texte de la chaîne de caractères doit être absolument placé entre guillemets :

A$="Atari ST"

Si vous essayez d'affecter une valeur numérique à une variable alphanumérique, vous obtenez le message d'erreur :

Types of values do not match

Vous obtiendrez d'ailleurs le même message d'erreur si vous essayez d'affecter une chaîne de caractères à une variable numérique :

A="TEST" (ERREUR)!

Le seul opérateur de calcul qui puisse être utilisé en liaison avec les chaînes de caractères, est le signe plus +. Ce signe permet de "concaténer" deux chaînes de caractères. Si vous définissez A$="LECTEUR DE " et B$="DISQUETTE",
A$+B$ donnera "LECTEUR DE DISQUETTE":

```
10 A$="LECTEUR DE ":B$="DISQUETTE"
20 LD$=A$+B$
30 PRINT LD$
40 END
```

La ligne 10 initialise les variables A$ et B$. La ligne 20 affecte le résultat de la concaténation de A$ et B$ à la variable LD$. La ligne 30 sort la chaîne ainsi obtenue.

Vous ne pouvez pas seulement concaténer différentes chaînes de caractères, vous pouvez également les comparer entre elles d'après leur place dans l'ordre alphabétique ou leur nombre de caractères. Nous reviendrons sur cette possibilité lorsque nous traiterons des instructions de comparaison (voir IF ... THEN). Notez simplement pour le moment que vous ne pouvez comparer entre elles des variables de différents types : vous ne pouvez comparer des valeurs numériques à des chaînes de caractères.

Le BASIC de l'Atari ST vous permet également de manipuler les chaînes de caractères. C'est aux instructions qui permettent ces manipulations que nous allons maintenant nous intéresser.

2.5.1 LEFT$

L'instruction LEFT$ permet de constituer une chaîne de caractères à partir d'une partie d'une chaîne déterminée. Pour plus de clarté, veuillez taper maintenant le programme suivant :

```
10 A$="COMPUTER"
20 B$=LEFT$(A$,1)
30 C$=LEFT$(A$,2)
40 D$=LEFT$(A$,3)
50 E$=LEFT$(A$,4)
60 F$=LEFT$(A$,5)
70 G$=LEFT$(A$,6)
80 H$=LEFT$(A$,7)
90 I$=LEFT$(A$,8)
100 PRINT B$:PRINT C$:PRINT D$:PRINT E$
110 PRINT F$:PRINT G$:PRINT H$:PRINT I$:PRINT A$
120 END
```

Lancez maintenant ce programme avec RUN. Vous obtenez :

```
C
CO
COM
COMP
COMPU
COMPUT
COMPUTE
COMPUTER
```

Cet exemple montre bien le fonctionnement de l'instruction LEFT$. La ligne 10 affecte à la variable A$ la chaîne de caractères "COMPUTER". La ligne 20 constitue une sous-chaîne de A$ en prenant le premier caractère de A$ en partant de la gauche. Cette sous-chaîne est affectée à la variable B$. La ligne 30 constitue également une sous-chaîne de A$ en prenant cette fois les deux premiers caractères de A$ en partant de la gauche. Cette sous-chaîne est affectée à la variable C$. Les lignes 40 à 90 fonctionnent de la même façon.

L'instruction LEFT$(A$,X) crée donc une chaîne avec les X premiers caractères de A$ en partant de la gauche. Les lignes 100 et 110

permettent de sortir les nouvelles chaînes constituées.
Nous avons cédé ici à la facilité qui consiste à écrire plusieurs instructions sur une même ligne parce que ces deux lignes restent malgré tout parfaitement compréhensibles.

Comme vous le voyez cette instruction peut être très utile pour obtenir des effets dans des jeux mais elle peut aussi déboucher sur des applications très sérieuses, notamment dans le domaine du traitement de données.

2.5.2 RIGHT$

Etudions maintenant l'instruction RIGHT$ dont les effets sont très proches de ceux de l'instruction LEFT$. En effet cette instruction RIGHT$ permet également de constituer une sous-chaîne d'une chaîne donnée, mais en prenant les caractères en partant de la droite de la chaîne de référence, au lieu de partir de la gauche. Si vous remplacez maintenant toutes les instructions LEFT$ de notre exemple précédent par des instructions RIGHT$, vous obtiendrez après avoir lancé le programme :

```
R
ER
TER
UTER
PUTER
MPUTER
OMPUTER
COMPUTER
```

Vous pouvez également modifier le programme à nouveau en faisant varier les paramètres de l'instruction RIGHT$ de 8 à 1 au lieu qu'ils varient de 1 à 8; vous obtenez alors l'image contraire. Ces exemples ont pour simple but de vous montrer le fonctionnement de cette instruction, de façon à ce que vous puissiez ensuite donner libre cours à votre fantaisie dans l'emploi de cette instruction.

2.5.3 MID$

Une des instructions les plus intéressantes de manipulation des chaînes de caractères est l'instruction MID$. Voici tout de suite un exemple d'application de cette instruction qui nous permettra de comprendre plus aisément sa fonction :

```
10 A$="SOCIETENATIONALEDESCHEMINSDEFERFRANCAIS"
20 B$=MID$(A$,1,7)
30 C$=MID$(A$,8,9)
40 D$=MID$(A$,20,12)
50 E$=MID$(A$,32,8)
60 PRINT A$
70 PRINT B$
80 PRINT C$
90 PRINT D$
100 PRINT E$
110 END
```

Lancez maintenant le programme et examinez le résultat. Vous voyez donc que l'instruction MID$ vous permet de constituer des sous-chaînes d'une chaîne de caractères, en prenant le nombre que vous voulez de caractères de cette chaîne à partir du caractère de cette chaîne que vous fixez :

MID$(A$,X,Y)

A$ est le nom de la chaîne de départ. X est la position dans la chaîne de départ du premier caractère à partir duquel doit être constituée la sous-chaîne et Y est le nombre de caractères qui doivent constituer cette sous-chaîne. C'est ainsi que l'instruction MID$ en ligne 20 crée à partir de :
"societenationaledescheminsdeferfrancais" une sous-chaîne de 7 caractères commençant par le premier caractère de cette expression. Dans ce cas, si le premier paramètre de cette instruction est égal à 1 vous voyez qu'on aboutit au même résultat qu'avec l'instruction LEFT$(A$,7).

Nous avons ensuite fait un usage plus normal de cette instruction à partir de la ligne 30 qui affecte par exemple à C$ la sous-chaîne de A$ composée de 9 caractères de A$ à partir du 8ème caractère de A$.
Notez que cette instruction est très puissante puisque vous pouvez utiliser comme paramètres aussi bien des variables ou des expressions arithmétiques que des constantes. Vous pouvez également utiliser cette instruction pour modifier le contenu d'une chaîne de caractères :

```
MID$(A$,7,1)="a"
```

transformera "societe" en "societa".

2.5.4 LEN(X$)

Avant de passer à une autre instruction, un petit exercice : comptez combien de caractères comporte notre expression A$("societenationaledescheminsdeferfrancais"). Si vous avez bien compté, vous devez en trouver 39.

Le seul but de ce petit exercice était de vous faire entrevoir l'intérêt de l'instruction que nous allons maintenant étudier, l'instruction LEN(X$) qui fournit la longueur d'une chaîne de caractères. Si vous entrez maintenant :

```
PRINT LEN(A$)
```

vous voyez apparaître 39. Notez que tous les caractères d'une chaîne de caractères sont pris en compte par LEN(X$), y compris les espaces.

2.5.5 VAL(X$)

L'instruction VAL(X$) est une autre instruction de maniement des chaînes de caractères qui vous permet de transformer toute chaîne de caractères en une expression numérique. Cette instruction vous permet surtout de transformer les nombres qui se trouvent pour une raison ou une autre sous forme d'une chaîne de caractères en une valeur numérique avec laquelle vous allez pouvoir effectuer des calculs. Si le premier caractère de la chaîne de caractère dont vous voulez connaître la valeur numérique avec cette instruction est un caractère qui ne peut être interprété comme un chiffre, la valeur obtenue sera 0. Si un caractère qui ne peut pas être interprété comme un chiffre se trouve à l'intérieur de la chaîne de caractères considérée, la valeur obtenue sera le nombre constitué par les chiffres placés avant ce caractère non numérique. Voici quelques exemples qui vous permettront de mieux comprendre le fonctionnement concret de cette instruction :

EXEMPLES

a)
```
10 A$="343.45"
20 A=VAL(A$)
30 PRINT A
```

Résultat :

343.45

b)
```
10 B$="D34.87F"
20 B=VAL(B$)
30 PRINT B
```

Résultat :

0

c)
```
10 C$="234FFC54"
20 C=VAL(C$)
30 PRINT C
```

Résultat :

234

d)
```
10 D$="33,21"
20 D=VAL(D$)
30 PRINT D
```

Résultat :

33

Entrez ces petits exemples dans votre ordinateur et essayez-les. L'exemple a) vous montre un cas où la chaîne de caractères peut être entiérement transformée en une expression numérique. La chaîne B$ commence par contre par un caractère non numérique et sa valeur est donc nulle. C$ montre une chaîne mixte dont seule la partie numérique placée avant un caractère non numérique sera prise en compte pour le calcul de la valeur de la chaîne. Le dernier exemple a simplement pour but de vous rappeler qu'en BASIC, contrairement aux mathématiques européennes, la virgule ne peut être interprétée comme un caractère numérique. C'est le point qui en tient lieu, comme dans notre premier exemple.

2.5.6 STR$(X)

L'instruction VAL(X$) est exactement symétrique à l'instruction STR$(X) qui convertit en effet toute valeur numérique en une chaîne de caractères pouvant donc être traitée comme telle avec les nombreuses instructions de maniement des chaînes de caractères que nous avons vues plus haut. Notez que le premier caractère de la chaîne ainsi créée ne sera pas le premier chiffre de la valeur numérique à convertir mais le signe de cette valeur numérique, donc un espace pour un nombre positif et le signe moins pour un nombre négatif.

EXEMPLES

a)
```
10 A=1234
20 A$=STR$(A)
30 PRINT A$
```

Résultat :

1234

b)
```
10 B=-1234
20 B$=STR$(B)
30 PRINT B$
```

Résultat :

```
-1234
```

Les chaînes créées par nos deux exemples comportent donc toutes deux 5 caractères. Comme c'est le cas de toutes les intructions BASIC, le paramètre de cette instruction peut être une variable ou une expression arithmétique.

2.5.7 INSTR

Voici maintenant une autre instruction importante pour manipuler les chaînes de caractères, l'instruction :

INSTR

qui vous permet de rechercher si une suite de caractères quelconque se trouve dans une chaîne de caractères :

INSTR(X,A$,B$)

A$ est la chaîne de caractères dans laquelle doit être recherchée la suite de caractères B$. X est la position du caractère de A$ à partir duquel doit commencer la recherche. Le résultat obtenu est la position de la suite de caractères B$ dans la chaîne A$. Ce résultat est nul si la chaîne recherchée n'a pas été trouvée. Voici un petit programme d'exemple :

```
10 A$="societenationaledescheminsdeferfrancais"
20 B$="nationale"
30 Z=INSTR(1,A$,B$)
40 PRINT Z
50 END
```

Lancez ce programme avec RUN et vous obtiendrez 8 comme valeur de Z. Notez que cette instruction recherche exactement la suite de caractères que vous avez indiquée. Ceci signifie donc, dans notre petit programme, que l'ordinateur n'aurait pas trouvé l'expression "Nationale" parce que "Nationale" ne figure pas avec une majuscule dans A$.

A RETENIR :

La suite de caractères que vous faites rechercher doit correspondre exactement à celle qui figure dans la chaîne que vous faites examiner. Faites attention aux majuscules et aux minuscules.

2.5.8 STRING$

Voici maintenant une instruction qui peut notamment vous permettre d'améliorer la présentation de vos programmes dont nous parlerons plus loin. Cette instruction :

STRING$

crée une chaîne de caractères composée du même caractère :

```
A$=STRING$(40,"*")      <RETURN>
PRINT A$                <RETURN>
```

Vous obtenez une ligne de 40 étoiles. Ceci peut vous permettre également de réaliser aisément des masques écran, mais c'est un point sur lequel nous reviendrons plus loin.

La limite est ici aussi de 255 caractères.

2.5.9 SPACE$

L'instruction :

SPACE$

équivaut à l'instruction STRING$(X," "). Elle crée donc une chaîne composée d'espaces :

A$=SPACE$(20)

créera une chaîne de 20 espaces. Cette instruction peut être utilisée pour positionner un texte sorti sur l'écran :

PRINT SPACE$(10);"Atari ST"

donnera :

<10 espaces> Atari ST

Voici maintenant quelques exercices pour vous permettre de consolider vos connaissances nouvellement acquises. Nous vous souhaitons comme toujours beaucoup de succès.

EXERCICES

1. Essayez sans les entrer dans l'ordinateur de dire quelle sera la différence sur l'écran entre les résultats des deux instructions suivantes :

PRINT SPC(5)"TEST1"TAB(15)"TEST2"

PRINT TAB(5)"TEST1"TAB(15)"TEST2"

a) La seule différence sera qu'avec la première instruction, TEST1 sera sorti un caractère plus à droite.

b) Avec la première instruction, 5 espaces sont produits avant que ne commence la sortie du premier mot. La sortie du second mot se fait après 15 autres espaces.
Avec la seconde instruction, 5 espaces sont également produits avant que ne commence la sortie du premier mot mais la sortie du second mot se fait après 10 autres espaces seulement.

c) Avec la première instruction, 5 espaces sont produits avant que ne commence la sortie du premier mot et la sortie du second mot se fait déjà après 10 autres espaces.
Avec la seconde instruction, 5 espaces sont également produits avant que ne commence la sortie du premier mot mais la sortie du second mot se fait seulement aprés 15 autres espaces.

2. Quelle expression obtient-on pour B$ en entrant la suite d'instructions suivantes :

A$="MOULIN A VENT" <RETURN>

B$=MID$(A$,1,1)+MID$(A$,8,1)+MID$(A$,4,3)<RETURN>

3. Quelle expression obtient-on pour A$ en entrant la suite d'instructions suivantes :

A$="ROTOR" <RETURN>

A$=LEFT$(A$,3)+RIGHT$(A$,2) <RETURN>

4.Quelle suite d'instructions permettra de créer à partir de A$="CHAPITEAU DE CIRQUE", B$="CHATEAU DE PITRE"?

2.6 L'EDITION DES PROGRAMMES

Avant que nous ne passions à la réalisation de programmes de taille plus importante, il nous faut nous intéresser aux instructions qui facilitent le travail de programmation. Le terme d' "édition" des programmes qui nous est plus ou moins imposé par l'origine anglaise du BASIC regroupe tout ce qui se rapporte à la modification d'un programme, qu'il s'agisse d'ajouter, de supprimer ou de modifier des lignes de programme, notamment pour remédier aux erreurs de syntaxe décelées par l'ordinateur. Nous supposons toutefois connu le maniement de l'éditeur (fenêtre d'édition). Si vous avez malgré tout encore des difficultés, nous vous invitons à vous reporter au chapitre approprié du manuel de votre Atari ST.

Nous avons déjà indiqué qu'il est recommandé de numéroter les lignes de programmes de 10 en 10. Vous pouvez pour cela utiliser l'instruction :

AUTO

Si vous entrez, par exemple, AUTO 10,10, vous voyez apparaître le nombre 10 et le curseur. L'ordinateur attend ainsi que vous entriez la ligne 10 du programme. La première valeur transmise à l'instruction AUTO détermine la première ligne de programme que vous voulez écrire. La seconde valeur détermine l'incrément, c'est-à-dire l'espacement entre les différents numéros des lignes consécutives. Vous pouvez mettre fin à la numérotation automatique avec les touches Control G.

Si vous écrivez un programme sans utiliser la numérotation automatique, vous risquez d'avoir besoin à un moment ou un autre de l'instruction :

RENUM

Lorsque vous développez un programme, il arrive fréquemment que vous ayez à rajouter des lignes, de sorte que la numérotation de votre programme se présente vite ainsi :

```
10 ...
12
15
19
20
21 ...
```

Ceci peut être très gênant si vous vous apercevez que vous avez maintenant besoin de rajouter une ligne entre 19 et 20 ou entre 20 et 21. Vous pouvez alors utiliser l'instruction RENUM qui renumérotera votre programme de 10 en 10. Dans l'exemple ci-dessus, votre programme se trouvera donc numéroté maintenant de 10 à 60. Cette instruction est très puissante puisqu'elle modifie automatiquement les numéros de ligne placés après les instructions de saut GOTO, GOSUB, etc... en fonction de la nouvelle numérotation.

Vous pouvez également renuméroter uniquement une partie de votre programme :

RENUM X,Y,Z

renumérotera de Z en Z les lignes à partir de la ligne numéro Y en donnant à la ligne numéro Y le numéro X :

RENUM 200,100,5

renumérotera de 5 en 5 les lignes à partir de la ligne numéro 200 en donnant à la ligne numéro 200 le numéro 100. L'instruction RENUM crée un fichier sous le nom de 'BASIC.WRK'. Veillez donc à ce que la disquette placée dans le lecteur de disquette ne soit pas protégée contre l'écriture.

Il peut arriver que vous ayez besoin de supprimer une partie seulement de votre programme (NEW vous permet en effet de le supprimer entièrement). Vous pouvez alors utiliser l'instruction :

DELETE

dont la syntaxe est :

DELETE 100-200

supprime les lignes 100 à 200,

DELETE -200

supprime toutes les lignes jusqu'à la ligne 200 et

DELETE 200-

supprime toutes les lignes à partir de la ligne 200.

Soyez cependant prudent dans l'utilisation de cette instruction car aucune demande de confirmation n'est prévue.

Si vous voulez tester un programme, vous pouvez le faire interrompre en n'importe quel endroit pour vérifier s'il a fonctionné correctement jusqu'ici en plaçant à cet endroit l'instruction :

STOP

Lorsque le programme rencontrera cette instruction, il s'interrompra en affichant le message :

Stop at line <numéro de ligne>

Vous pouvez alors faire repartir le programme à partir de cet endroit avec l'instruction :

CONT

L'instruction CONT est très différente de l'instruction RUN qui lance un programme après avoir annulé toutes les variables et qui ne permet donc pas de poursuivre l'exécution normale du programme, même si vous l'utilisez en indiquant le numéro de la ligne à laquelle le programme doit reprendre. Vous ne pouvez pas, toutefois, relancer un programme avec CONT si le programme a été interrompu à la suite d'un message d'erreur.

Il est parfois très intéressant de pouvoir suivre le déroulement du programme en connaissant au fur et à mesure les numéros des lignes que le programme est en train d'exécuter. Ceci permet par exemple de faire des comparaisons entre le fonctionnement du programme et votre organigramme. Vous disposez pour ce faire de l'instruction :

TRON

Une fois entrée, cette instruction (TRace ON) affiche, pendant le déroulement du programme, le numéro de chaque ligne avant de l'exécuter. L'instruction :

TROFF

(TRace OFF) met fin à cette fonction.

L'instruction :

TRACE

indique à tout moment la ligne du programme que celui-ci est en train de traiter. En indiquant un numéro de ligne, vous pouvez également ne faire afficher que certaines lignes. Par exemple,

TRACE 40

n'indiquera que la ligne 40.

Les combinaisons possibles avec TRACE sont les mêmes qu'avec les instructions DELETE ou LIST.

Avec l'instruction :

UNTRACE

vous désactivez cette fonction.

Si vous voulez savoir le contenu des variables dans les différentes lignes de programme, vous pouvez utiliser à cet effet l'instruction

FOLLOW

FOLLOW A vous indiquera par exemple le contenu de la variable A ainsi que le numéro de ligne correspondant.

En entrant :

UNFOLLOW

vous mettez fin à cette instruction.

L'instruction :

NEW

supprime le programme se trouvant en mémoire. Vous avez déjà rencontré cette instruction dans vos exercices. Nous ne la rappelons que pour être complet.

L'instruction :

LIST

vous permet de sortir sur l'écran tout ou partie du programme.

Pour définir les lignes que vous voulez faire sortir sur l'écran, vous pouvez employer exactement la même syntaxe que celle de l'instruction DELETE. LLIST vous permet de sortir le programme sur l'imprimante.

Deux instructions vous permettent de supprimer les variables créées :

CLEAR

supprime toutes les variables et tous les tableaux. Si vous ne connaissez pas encore le concept de "tableau", ne vous inquiétez pas, nous l'expliquerons plus loin. Cette instruction vous permet donc d'annuler en n'importe quel emplacement de votre programme toutes les variables créées.

Vous pouvez aussi ne supprimer qu'un tableau ou que des tableaux, vous pouvez utiliser l'instruction :

ERASE

ERASE A

supprimera le tableau créé avec DIM A(20). Vous pouvez ainsi libérer de la place en mémoire et vous pouvez également, ce qui est parfois très utile, procéder à un second dimensionnement d'un tableau au cours du même programme.

Ces instructions d'édition constituent une aide très appréciable pour la réalisation et la correction de vos programmes.

2.7 LES FENETRES ECRAN

Vous pouvez fermer, ouvrir et supprimer les différentes fenêtres écran (Edit, List, Output et Command) y compris en BASIC. Voici donc une brève description des différentes instructions qui vous le permettent. Les valeurs suivantes sont affectées aux différentes fenêtres :

0 = fenêtre Edit
1 = fenêtre List
2 = fenêtre Output
3 = fenêtre Command

L'instruction :

FULLW 2

fixe la fenêtre de sortie sur les dimensions de l'écran complet.

Si vous voulez supprimer le contenu d'une fenêtre, utilisez pour cela l'instruction :

CLEARW

CLEARW 3 supprime par exemple le contenu de la fenêtre Command.

L'instruction :

OPENW

ouvre une fenêtre qui avait été fermée auparavant.

L'instruction :

CLOSEW

vous permet de fermer une fenêtre.
La fenêtre correspondante disparaîtra totalement du Desktop.

ATTENTION !

Si vous fermez TOUTES les fenêtres, l'ordinateur ne réagira plus à aucune entrée. Vous serez alors obligé de lancer le système à nouveau.

Nous en avons fini avec ce chapitre d'introduction à la programmation en BASIC. Le chapitre suivant abordera des structures de programme plus complexes ainsi que la programmation des boucles.

CHAPITRE 3

STRUCTURES DE PROGRAMME PLUS COMPLEXES

Nous n'avons abordé jusqu'ici que les programmes à déroulement linéaire. C'est maintenant aux sauts ou aux branchements à l'intérieur d'un programme que nous allons nous intéresser.
L'inconvénient des programmes à déroulement linéaire réside en effet dans le fait que le programme est exécuté une fois de la première ligne à la dernière et qu'il faut ensuite le relancer avec l'instruction RUN si on souhaite continuer à l'utiliser. D'autre part, comme il n'y a pas de branchements à l'intérieur du programme, on est obligé de faire exécuter chaque fois l'intégralité du programme, même si dans certains cas on souhaiterait pouvoir utiliser plusieurs fois de suite telle ou telle partie du programme. Si tous les programmes étaient donc à déroulement linéaire, l'informatique ne pourrait résoudre que des problèmes très simples, tels que ceux que nous avons jusqu'ici résolus.

3.1 SAUTS INCONDITIONNELS

La forme la plus simple de branchement à l'intérieur d'un programme, est le saut ou branchement inconditionnel avec l'instruction GOTO que nous allons illustrer à l'aide d'un programme d'exemple.

Cette instruction ordonne à l'ordinateur d'interrompre l'exécution linéaire (ligne par ligne dans l'ordre croissant des numéros de ligne) du programme pour sauter à la ligne dont le numéro est placé à la suite de l'instruction GOTO. Il s'agit d'une instruction de saut inconditionnel car le programme exécute dans tous les cas le branchement voulu par l'instruction GOTO.

L'inconvénient des branchements ou sauts inconditionnels est toutefois qu'ils ne permettent de réaliser que des "boucles infinies": une fois le programme lancé, il ne s'arrêtera plus à moins que vous n'éteigniez votre ordinateur ou que vous n'utilisiez la touche Control G pour l'interrompre.

Nous allons maintenant utiliser l'instruction GOTO pour développer un des exemples que nous avons déjà donnés. Souvenez-vous de l'exercice dans lequel nous vous demandions de réaliser un programme calculant le poids idéal d'une personne.

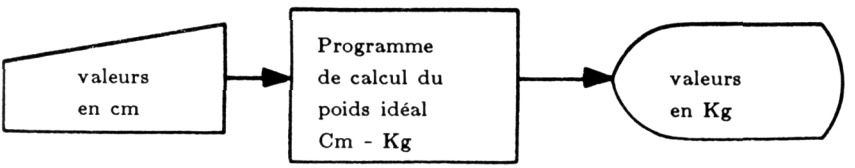
Essayons maintenant de développer ce programme de façon à en faire un petit gag pour vos petites soirées: chacun de vos invités pourra connaître son poids idéal. Sans l'instruction GOTO, ce programme devrait être relancé pour chaque invité. C'est pourquoi nous allons ajouter avant la dernière ligne de la première version de notre programme une instruction GOTO qui imposera à l'ordinateur de sauter à nouveau en début de programme après avoir donné le poids idéal d'un invité :

```
10 INPUT"ENTREE TAILLE EN CM";CM
20 REM CALCUL POIDS IDEAL
30 IDEAL=(CM-100)-(CM-100)/100*10
40 REM SORTIE
50 PRINT"VOTRE POIDS IDEAL EST DE";IDEAL;"KG"
60 REM SAUT INCONDITIONNEL AVEC GOTO
70 GOTO 10
80 END
```

Le calcul est effectué dans une seule ligne de programme. Vous voyez que le programme, après avoir sorti le poids idéal d'une personne en ligne 50, rencontre en ligne 70 une instruction de branchement inconditionnel. Il saute alors en ligne 10 pour permettre l'entrée des paramètres correspondant à une autre personne. La ligne 80 END est d'ailleurs parfaitement inutile car elle ne pourra jamais être atteinte par le programme puisqu'elle est "court-circuitée" par le saut en ligne 70.

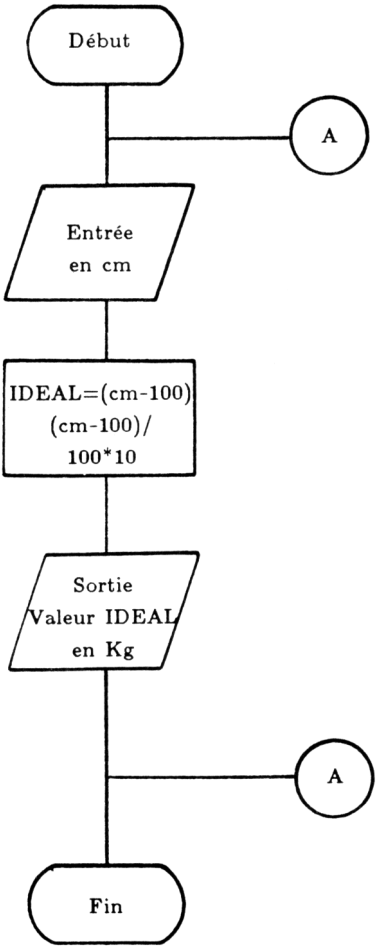
Le plan de flux des données n'est pas affecté par cette nouvelle instruction; il se présenterait de toute façon ainsi :

PLAN DE FLUX DE DONNEES
POUR LE CALCUL DU POIDS IDEAL



Par contre l'organigramme de notre programme est modifié ainsi que vous pouvez le voir ci-après :

ORGANIGRAMME DU PROGRAMME POIDS IDEAL



Vous voyez que nous avons utilisé un nouveau symbole, le connecteur, qui permet d'éviter de tracer de grands traits à travers la page pour marquer les sauts du programme.

Un premier connecteur figure avant la fin du programme pour indiquer qu'il y a là un saut du programme (GOTO), un second se trouve après le début du programme pour indiquer où aboutit ce saut du programme. les deux connecteurs sont reliés entre eux par la lettre A qui leur est commune. Nous aurions pu également nous passer ici du symbole de fin qui est court-circuité par le premier connecteur comme la ligne 80 END est court-circuitée par la ligne 70 GOTO 10. Vous voyez donc que l'instruction GOTO ne permet à elle seule que de réaliser des boucles sans fin.

3.2 SAUTS CONDITIONNELS

La puissance des ordinateurs vient de leur faculté d'effectuer des décisions logiques ou des comparaisons. Un ordinateur peut en effet tester si une variable est inférieure ou supérieure à 0 et ensuite effectuer ou ne pas effectuer un branchement suivant le résultat de ce test. L'instruction IF ... THEN permet par exemple d'effectuer de tels branchements conditionnels en fonction du résultat d'une comparaison.

3.2.1 IF...THEN...ELSE

Lorsque l'ordinateur rencontre une instruction IF ... THEN ... ELSE (si ... alors ... sinon), il examine d'abord la condition qui a été placée à la suite de IF. Si cette condition est VRAIE, c'est-à-dire si elle est remplie, l'ordinateur exécute la ou les instructions placées à la suite de THEN. Si par contre cette condition est FAUSSE, c'est-à-dire si elle n'est pas remplie, l'ordinateur exécute la ou les instructions placées à la suite de ELSE et en tout cas il n'exécute pas les instructions placées à la suite de THEN. Le mot ELSE est facultatif.

Le mot IF peut être suivi d'opérateurs logiques, de chaînes de caractères, de variables, de comparaisons et de nombres ou de combinaisons de ces divers éléments.

Le mot THEN est suivi le plus souvent du numéro d'une ligne à laquelle le programme doit sauter. Il peut être aussi suivi d'affectations de valeurs à des variables ou de sauts à des sous-programmes.

Voici maintenant un exemple simple de l'emploi de l'instruction IF ... THEN :

```
10 INPUT "Entrez un nombre";X
20 IF X<0 THEN 50
30 IF X>0 THEN 70
40 IF X=0 THEN 90
50 PRINT "Ce nombre est plus petit que zéro"
60 GOTO 100
70 PRINT "Ce nombre est plus grand que zéro"
80 GOTO 100
90 PRINT "Ce nombre est égal à zéro"
100 END
```

Ce programme vous permet d'entrer un nombre quelconque, après quoi il vous indique si ce nombre est plus petit ou plus grand que zéro ou bien égal à zéro. Bien entendu, vous le savez vous-même et le seul intérêt de ce programme est de vous montrer comment l'ordinateur exécute l'instruction IF ... THEN :
Si vous entrez maintenant un nombre supérieur à 0, l'ordinateur rencontre d'abord la ligne 20. Il teste alors si le nombre que vous avez entré est inférieur à 0. Comme cette condition n'est pas remplie, il ne saute pas en 50 mais passe à la ligne suivante. Il teste alors si le nombre que vous avez entré est supérieur à 0. Comme cette condition est remplie, il exécute l'instruction placée à la suite de THEN et saute en 70. La ligne 70 affiche alors le message indiquant que le nombre entré est supérieur à 0.
L'ordinateur rencontre ensuite en ligne 80 l'instruction de branchement inconditionnel GOTO 100 et il saute à la ligne 100 qui termine le programme. Si vous voulez au contraire que le programme tourne continuellement, il vous suffit de remplacer l'instruction END de la ligne 100 par l'instruction GOTO 10.

Après ce programme d'exemple simple, nous allons aborder maintenant un programme déjà un peu plus compliqué. Vous connaissez certainement le jeu où quelqu'un doit penser à un nombre qu'une autre personne doit essayer de deviner. Après chaque question de la personne qui doit deviner, on doit dire si le nombre proposé est plus grand, plus petit ou égal au nombre à découvrir. Nous allons maintenant réaliser ce jeu sur notre ordinateur. Entrez donc le programme suivant :

```
10 REM DEVINER UN NOMBRE
20 CLEARW 2:PRINT
30 PRINT "Entrez deux nombres correspondant "
40 PRINT "à la limite inférieure et à la limite supérieure."
50 INPUT"Limite inférieure";inf
60 INPUT"Limite supérieure";sup
70 Z=INT((sup+1)*RND(1))+inf
80 INPUT"PROPOSEZ UN NOMBRE";sz
90 IF sz<z THEN 120
100 IF sz>z THEN 140
110 IF sz=z THEN 160
120 PRINT"Le nombre à trouver est plus grand"
130 GOTO 80
140 PRINT"Le nombre à trouver est plus petit"
150 GOTO 80
160 CLEARW 2:PRINT"Bravo! vous avez trouvé!"
170 PRINT"Voulez-vous rejouer oui/non"
180 INPUT A$
190 IF A$="oui" THEN 20
200 END
```

Les premières lignes n'ont pas besoin de commentaire. Les lignes 30 à 60 demandent au joueur d'entrer les limites entre lesquelles doit être compris le nombre à deviner. La ligne 70 détermine dans les limites qui viennent d'être définies le nombre à trouver au moyen de la fonction RND. Si vous ne comprenez pas bien la logique de la ligne 70, reportez-vous au chapitre où nous avons traité des nombres aléatoires.

La ligne 80 vous demande maintenant d'entrer un nombre. Ce nombre est maintenant comparé en lignes 90 à 110 au nombre qui a été tiré au sort et que contient la variable Z. Suivant que le nombre que vous avez entré est plus grand, plus petit ou égal au nombre à trouver, le programme saute à la ligne correspondante et le programme continue. Si vous avez trouvé le nombre à découvrir, le programme saute en ligne 160 et on vous demande en ligne 170 si vous voulez rejouer. Si vous entrez Oui, la condition posée en ligne 190 se trouve remplie et le programme recommence, sinon le programme est terminé.

L'instruction IF ... THEN en lignes 90 à 110 sert donc à comparer le nombre entré par le joueur (SZ) et le nombre tiré au sort par l'ordinateur (Z). En ligne 190 par contre, l'instruction IF ... THEN sert à comparer une variable alphanumérique. Notez bien que, pour que la condition en ligne 190 soit vraie, il faut absolument que les deux chaînes comparées soient rigoureusement identiques. Bien entendu, si vous entrez 'yes', le programme sera terminé malgré tout. Si vous voulez rejouer, vous devez absolument entrer 'oui'.

L'instruction IF ... THEN nous permet maintenant également de réaliser des "boucles dirigées", c'est-à-dire non plus simplement des boucles sans fin que le programme parcourt indéfiniment mais des boucles que le programme parcourra jusqu'à ce qu'une condition déterminée ait été remplie. Le petit programme d'exemple suivant vous montre comment programmer une boucle dirigée. Notre programme sort sur l'écran la table de 3 :

```
10 A=3
20 PRINT A
30 A=A+3
40 IF A>30 THEN 60
50 GOTO 20
60 END
```

La ligne 10 initialise la variable A avec la valeur 3. La ligne 30 crée ce qu'on appelle un compteur qui ajoute toujours 3 au contenu actuel de la variable A. La ligne 40 teste si A a déjà dépassé 30. Tant que A est inférieur ou égal à 30, le programme continue avec l'instruction GOTO en ligne 50. Nous avons ainsi réalisé une boucle qui sera parcourue exactement 10 fois par le programme. Nous disposons ainsi d'un moyen de créer des boucles qui ne soient parcourues qu'un nombre déterminé de fois.

L'exemple ci-dessus se prête bien à l'emploi du mot ELSE qui va nous permettre ici de faire l'économie de la ligne 50 :

```
10 A=3
20 PRINT A
30 A=A+3
40 IF A>30 THEN 60 ELSE 20
60 END
```

Si vous lancez cette nouvelle version du programme, vous constaterez que vous obtenez exactement le même résultat qu'avec la première version : tant que, lors des 10 premiers parcours, A est inférieur à 30 et que la condition aprés IF est fausse, c'est l'instruction placée après le mot ELSE qui est exécutée. Le programme saute alors à la ligne 20. Dès que A a atteint la valeur 33, le programme est terminé en ligne 60. Nous avons laissé à la ligne 60 son numéro pour bien mettre en évidence la suppression de la ligne 50. Le mot ELSE vous permet donc d'améliorer la présentation de vos programmes et d'économiser des lignes de programme et donc de la place en mémoire.

3.2.2 ETIQUETTES DE SAUT / LABELS

Le BASIC de l'Atari ST vous donne la possibilité de définir des étiquettes ou labels. Il est ensuite possible de sauter à ces labels avec GOTO ou GOSUB. Vous n'êtes plus ainsi obligé d'avoir en tête toutes les lignes de programme des différentes routines car au lieu de :

50 GOTO 250

vous pouvez entrer simplement :

50 GOTO expression

La ligne 250 devra alors se présenter ainsi :

250 expression :

Le nom de l'étiquette ou label doit se terminer par un double point. La définition des étiquettes vous permet de donner des noms aux différentes parties de vos programmes et qui rendra ces derniers plus lisibles. Si vous voulez appeler par exemple dans un programme une routine de tri, vous n'êtes pas obligé de rechercher péniblement le numéro de ligne, il vous suffit d'entrer :

GOTO Trier

ou

GOSUB Trier

Les noms des labels ne doivent pas commencer par un chiffre mais toujours par une lettre.

Voici maintenant à nouveau quelques exercices pour consolider vos connaissances. Bonne chance !

EXERCICES

1. Ecrivez un programme (théorique) qui calcule suivant votre revenu annuel un impôt de 33 ou de 51%. La limite sera de 50 000F : les revenus supérieurs à 50 000F seront imposés suivant un taux de 51%, les revenus inférieurs de 33%. Le résultat doit être présenté par un texte d'accompagnement.

2. Ecrivez un programme calculant la somme des nombres de 1 à 100.

3. Ecrivez un programme qui sorte 6 nombres aléatoires compris entre 1 et 49.

4. Quels nombres seront sortis par ce programme ? Essayez bien sûr de répondre avant d'entrer ce programme.

```
10 A=7
20 A=A+5:Z=Z+1
30 IF Z<9 THEN 20
40 PRINT A,Z
50 END
```

5. Ecrivez un programme qui vous permette de faire rechercher une sous-chaîne quelconque dans une chaîne de caractères quelconque. Testez votre programme en entrant INFORMATIQUE comme chaîne de caractères et FORMAT comme sous-chaîne à rechercher. La difficulté particulière de cet exercice résidera dans le fait que vous devrez résoudre cet exercice sans utiliser l'instruction INSTR. Il s'agira donc pour vous d'écrire une routine en BASIC qui remplace l'instruction INSTR.

3.2.3 FOR ... TO ... NEXT

Nous avons déjà créé des boucles, au moyen de l'instruction IF ... THEN. Le principe consistait à créer un compteur dont la valeur était augmentée ou diminuée lors de chaque parcours de la boucle. On testait à certains endroits du programme la valeur atteinte par le compteur, et suivant le résultat négatif ou positif du test, le programme sautait à nouveau au début de la boucle ou bien sortait de la boucle. La programmation d'une boucle telle que nous l'avons vue jusqu'ici est donc relativement complexe alors que les boucles sont un dispositif important dans tout programme qui ne se cantonne pas dans les limites du déroulement linéaire. Vous devez donc certainement vous douter déjà que le BASIC doit offrir une solution plus pratique pour arriver au même résultat. Et en effet l'instruction FOR ... TO ... NEXT permet de programmer les boucles beaucoup plus aisément comme le montre ce petit programme d'exemple :

```
10 REM SORTIE DES DIX PREMIERS CARRES
20 CLEARW 2:PRINT
30 PRINT"Sortie des 10 premiers carres"
40 FOR I=1 TO 10
50 PRINT"Le carré de";I;"est";I*I
60 NEXT I
70 PRINT"Fin"
```

Entrez ce programme et lancez-le. L'instruction FOR I=1 crée un compteur I auquel elle affecte la valeur de départ 1. Cette valeur de départ sera augmentée (incrémentée) de 1, jusqu'à ce qu'elle dépasse la valeur finale placée après TO. Ici, 10. Toutes les instructions placées entre FOR ... TO ... et l'instruction NEXT qui ferme la boucle seront exécutées, tant que la valeur du compteur I ne dépassera pas la valeur finale. Les valeurs initiale et finale peuvent être des nombres, des variables ou des expressions arithmétiques. Quelques exemples :

```
10 A=10:B=20
20 FOR Z=A TO B
30 PRINT Z;
40 NEXT Z
50 END
```

La ligne 10 initialise ici d'abord les variables A et B. La ligne 20 crée maintenant une boucle en utilisant ces deux variables comme valeur de départ et valeur finale. La ligne 30 sort les différentes valeurs de Z, tant que le compteur Z n'est pas supérieur à 20. Vous pouvez d'ailleurs vérifier en mode direct qu'une fois le programme terminé, Z aura bien une valeur supérieure à 20 :

```
PRINT Z
```

donnera : 21. Après avoir utilisé des variables pour définir la boucle, voici maintenant un exemple avec des expressions arithmétiques :

```
10 A=10:B=15:C=5
20 FOR Z=A TO A+B-C
30 PRINT Z;
40 NEXT Z
50 END
```

Ce programme revient exactement au même que l'exemple précédent mais la valeur finale est calculée à partir de l'expression A+B-C.

Si vous voulez que la boucle ait un autre pas d'incrémentation que 1, il faut définir ce pas avec l'instruction STEP. Voici par exemple un programme qui sort tous les nombres pairs entre 2 et 20:

```
10 REM NOMBRES PAIRS DE 2 A 20
20 FOR I=2 TO 20 STEP 2
30 PRINT I
40 NEXT I
50 END
```

La valeur de départ, la valeur finale ainsi que le pas d'incrémentation (ou de décrémentation) peuvent être également des nombres négatifs ou des fractions numériques. Voici par exemple un programme de compte à rebours :

```
10 REM COMPTE A REBOURS
20 FOR I=20 TO 0 STEP -1
30 PRINT I
40 NEXT I
50 END
```

Si vous lancez ce programme, vous allez constater que les chiffres défilent très rapidement alors qu'un vrai compte à rebours devrait compter les secondes. Nous allons, pour rendre effective la fonction de compte à rebours de notre programme, imbriquer des boucles:

```
10 REM compte a rebours
20 FOR I=20 TO 0 STEP -1  ----------+
30 PRINT I                         +
40 FOR Z=0 TO 1000        ---------+ +
50 REM boucle de temporisation     + +
60 NEXT Z                 --------- + +
70 NEXT I                  ---------- +
80 END
```

Entrez ce programme (sans les caractères graphiques que nous avons ajoutés pour marquer les boucles) et lancez-le: vous voyez que le compte à rebours marque presque exactement les secondes. Nous avons pour arriver à ce résultat utilisé une boucle de temporisation en lignes 40 à 60. Les boucles de temporisation sont utilisées très souvent, par exemple pour permettre à l'utilisateur de lire un texte avant que le programme n'affiche la page suivante sur l'écran. Examinons le fonctionnement des boucles imbriquées :

La première boucle est ouverte en ligne 20 avec I=20. La ligne 30 sort la valeur actuelle du compteur de la première boucle I. La ligne 40 ouvre la seconde boucle dont le NEXT est en ligne 60.

Cette boucle est d'abord parcourue entièrement, c'est-à-dire jusqu'à ce que Z soit supérieur à la valeur finale 1000, avant que la première boucle ne termine son premier parcours et ne puisse commencer son deuxième.

Les boucles imbriquées augmentent considérablement la puissance de vos programmes mais vous devez faire attention à les manipuler correctement et à ne pas réaliser de boucles croisées. Il doit toujours y avoir une hiérarchie précise entre les différentes boucles imbriquées et la première boucle ouverte doit toujours être la dernière fermée et la dernière ouverte doit toujours être la première fermée. Voici par exemple ce qu'il ne faut jamais faire :

FAUX

```
10 FOR I=1 TO 20  ----------------  +
20 PRINT I                          +
30 FOR Z=1 TO 10  ----------------+ +
40 PRINT Z                        + +
50 NEXT I         ----------------  +- +
60 PRINT I,Z                      +
70 NEXT Z         ----------------  +
```

FAUX

Si vous avez imbriqué plusieurs boucles et que vous voulez les fermer l'une après l'autre sans intercaler d'instructions entre les fins des différentes boucles, vous ne devez pas obligatoirement utiliser un NEXT spécial pour chaque instruction. Il suffit d'une seule instruction NEXT suivie des noms des différents compteurs de boucle, placés dans le bon ordre et séparés entre eux par une virgule :

```
10 FOR I=1 TO 10
20 FOR Z=1 TO 10
30 PRINT I;Z
40 NEXT Z,I
50 END
```

Vous voyez que nous avons fermé deux boucles avec une seule instruction NEXT mais que nous avons aussi respecté la règle qui veut que la dernière boucle ouverte (Z) soit fermée avant la première boucle ouverte (I).

Une autre faute à éviter lorsqu'on utilise des boucles dans un programme est de sauter à l'intérieur d'une boucle. Il est en effet fréquent qu'une boucle se compose de plusieurs lignes de programme et il est tout à fait possible de sauter directement à une de ces lignes. Mais lorsque l'ordinateur rencontrera l'instruction NEXT qui ferme la boucle, il affichera le message d'erreur :

You are trying to jump into a loop at line <numéro de ligne>

Si le programme n'est pas passé par l'instruction FOR qui ouvrait la boucle, il ne peut en effet pas savoir où il doit retourner quand il rencontre ce NEXT. Il s'agit d'ailleurs au fond de la même faute que celle qui consiste à réaliser des boucles croisées. Il faut toujours se rappeler qu'une boucle doit être parcourue entièrement, ou ne pas être parcourue. C'est là que se manifeste l'intérêt des organigrammes qui doivent normalement permettre d'éviter que de telles erreurs se produisent en les rendant beaucoup plus tangibles et évidentes.

Il faut d'autre part penser, si la valeur de départ que vous définissez est supérieure à la valeur finale, que vous devez indiquer un pas d'incrémentation négatif (=décrémentation). Sinon la boucle ne sera pas parcourue une seule fois :

```
10 FOR A=5 TO 1
20 PRINT A
30 NEXT A
40 END
```

On a ici oublié en ligne 10 d'indiquer le pas avec STEP -1. La valeur A ne sera pas imprimée une seule fois puisque A est dès le départ supérieur à la valeur finale 1.
Le programme ci-dessus est l'exemple d'une erreur de programmation mais vous pouvez utiliser le même principe pour provoquer un arrêt anticipé d'une boucle en affectant au compteur pendant le parcours d'une boucle une valeur supérieure à la valeur finale, si certaines conditions se sont produites qui rendent inutiles d'autres parcours. Vous pouvez également utiliser comme paramètres de l'instruction FOR ... TO des variables de façon à pouvoir obtenir des boucles de différentes longueurs. Voici d'abord un exemple d'interruption anticipée d'une boucle :

```
10 REM AUGMENTATION BRUSQUE
DE LA VALEUR DU COMPTEUR
20 FOR A=0 TO 20
30 PRINT A
40 IF A=12 THEN A=20
50 NEXT A
60 END
```

Vous voyez que cette boucle qui devait normalement afficher les nombres de 0 à 20 n'imprime que les nombres de 0 à 12 à cause de la ligne 40 qui met fin à la boucle dès que A vaut 12 en affectant 20 à la variable de compteur A. Retenez cette possibilité qui peut vous être utile un jour, bien qu'elle ne soit pas très souvent employée. Par contre, la possibilité suivante qui consiste à créer des boucles à "géométrie variable" est très utilisée, notamment dans les programmes de gestion de fichier pour rechercher par exemple certaines combinaisons de caractères :

```
10 INPUT"ENTREZ UN MOT";A$
20 FOR A=1 TO LEN(A$)
30 PRINT LEFT$(A$,A)
40 NEXT A
50 FOR A=LEN(A$) TO 1 STEP -1
60 PRINT RIGHT$(A$,A)
70 NEXT A
80 END
```

Entrez ce programme, lancez-le et entrez votre nom. Vous voyez que c'est ici la longueur du mot entré qui détermine le nombre de parcours de chaque boucle.

Essayons maintenant de résumer tout ce que nous avons appris sur les boucles FOR ... TO ... NEXT :

1. A chaque instruction FOR ... TO ... doit répondre une instruction NEXT. Une seule instruction NEXT peut fermer plusieurs boucles à la fois si elle est suivie des variables-compteurs de ces différentes boucles, placées dans l'ordre inverse de l'ordre d'ouverture des boucles correspondantes et séparées entre elles par une virgule.

2. On ne doit jamais sauter à l'intérieur d'une boucle sous peine de provoquer l'interruption du programme avec sortie d'un message d'erreur.

3. Si la valeur de départ est supérieure à la valeur finale du compteur d'une boucle, il faut indiquer le pas négatif de la boucle (avec STEP -x).

4. Une boucle FOR ... NEXT sera parcourue tant que la valeur de la variable-compteur ne sera pas supérieure à la valeur finale.

Ces règles ne s'appliquent qu'au BASIC de l'Atari ST et ne doivent donc pas être généralisées hâtivement.

Les différents dialectes BASIC existants divergent en effet quelque peu dans le traitement des boucles FOR ... NEXT.

3.2.4 WHILE ... WEND

La combinaison des instructions **WHILE ... WEND** vous offre un autre moyen particulièrement puissant et plus souple que FOR ... TO ... NEXT de réaliser des boucles. Vous n'avez pas en effet à définir d'incrément fixe ou de pas définitif.

Vous pouvez ouvrir une boucle avec WHILE que vous devez refermer avec WEND. L'expression logique qui est placée à la suite de WHILE est testée lors de chaque parcours de la boucle. Tant que cette expression est VRAIE, la boucle est parcourue jusqu'à WEND. Si cette expression n'est plus VRAIE, le programme saute à l'exécution des instructions placées aprés WEND. Vous pouvez par exemple faire interrompre une telle boucle par pur hasard, comme le montre l'exemple suivant :

```
10 WHILE A<100
20 A=INT(101*RND(1))
30 Z=Z+1
40 WEND
50 PRINT A,Z
60 END
```

Cette boucle sera parcourue tant que A sera plus petit que 100. La variable 100 reçoit sa valeur de la fonction aléatoire RND. Si A reçoit la valeur 100, le programme saute en ligne 50 et sort les valeurs de A et de Z. Le compteur Z vous indique ainsi combien de fois la boucle a été parcourue.

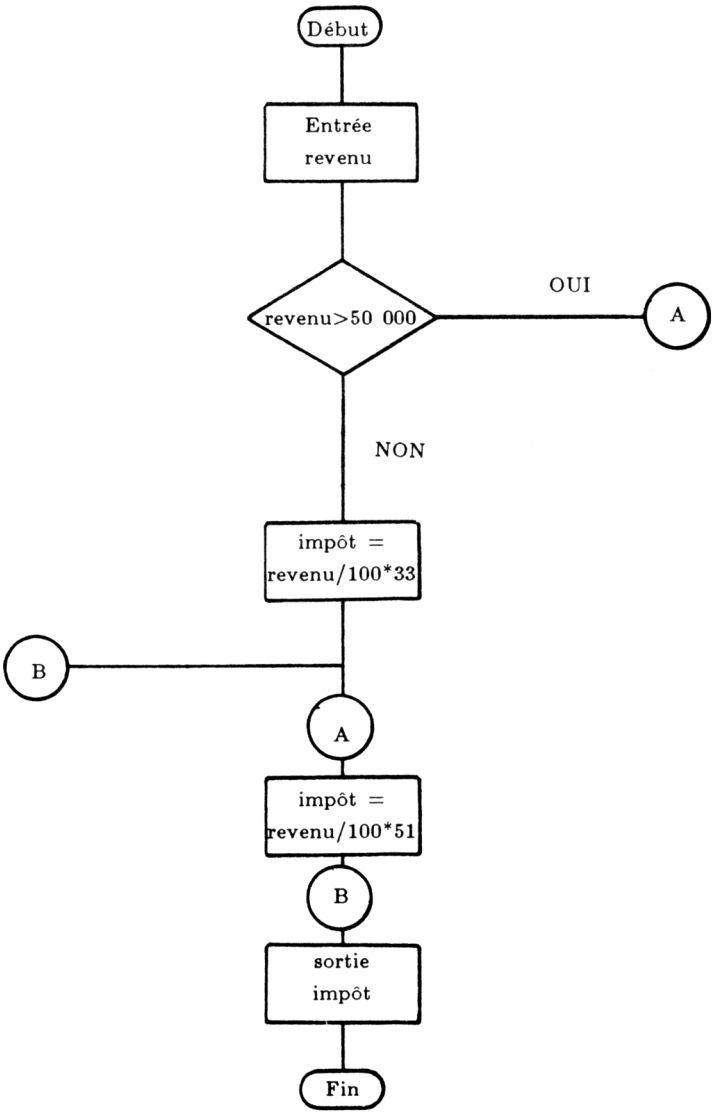
Voici maintenant les règles générales qu'on peut définir pour l'utilisation des boucles WHILE ... WEND :

1. Si vous connaissez d'avance le nombre de parcours de boucle nécessaires, il vaut mieux utiliser la boucle FOR ... TO ... NEXT.

2. Si vous ne connaissez pas d'avance le nombre de parcours de boucle, il vaut mieux utiliser les boucles avec IF ... THEN ou WHILE ... WEND.

Maintenant que nous avons étudié les branchements inconditionnels et les branchements conditionnels ainsi que la technique et les instructions de boucle, il nous manque encore le moyen de représenter ces structures de programmation dans nos organigrammes. C'est le losange qui est le symbole utilisé pour marquer un branchement logique (conditionnel) dans un organigramme. Voyons par exemple comment pourrait se présenter l'organigramme de notre programme de calcul du taux d'imposition :

ORGANIGRAMME DU PROGRAMME DE CALCUL DU TAUX D'IMPOSITION

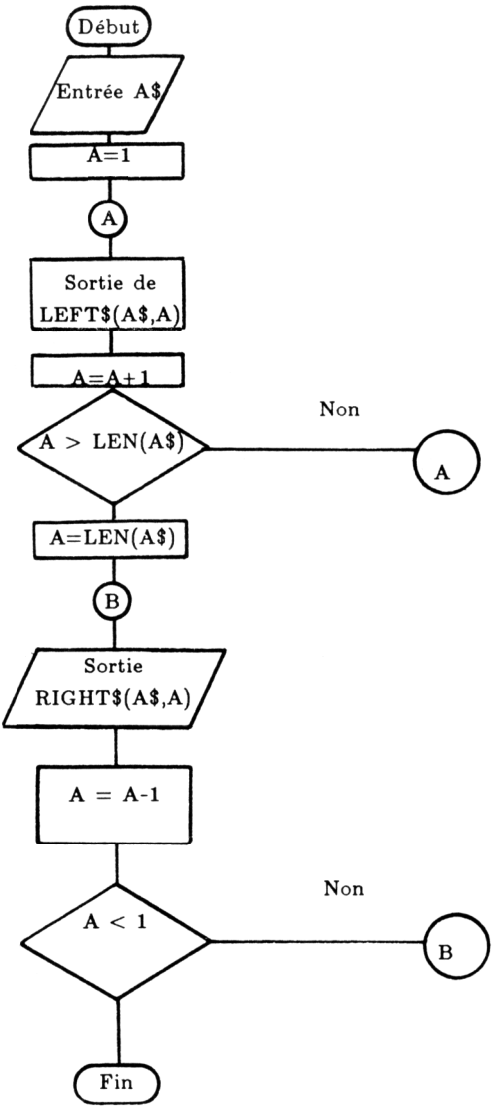


Le losange est donc le symbole d'un branchement logique : il comporte en effet une branche oui et une branche non. Si la condition figurant à l'intérieur du losange est remplie, le programme suit donc la branche oui qui le mène au connecteur A par lequel il saute au deuxième connecteur A. Si la condition n'est pas vérifiée, le programme poursuit son déroulement normal (linéaire) par la branche non. Dans notre exemple le branchement conditionnel se fait donc à travers la branche oui mais un branchement conditionnel peut aussi bien passer par la branche non. Le connecteur de saut A amène le programme au calcul de l'impôt avec un taux de 51 pour cent. L'impôt est ensuite sorti.

Si la condition n'est pas remplie l'impôt est calculé avec un taux de 33 pour cent. Le programme arrive alors au connecteur de saut B qui marque ici un branchement inconditionnel. Il faut veiller à ce que le connecteur de saut B soit bien placé avant le deuxième connecteur A, sinon il y aura une erreur de logique dans notre organigramme.

Cet organigramme nous montre comment représenter l'instruction IF ... THEN dans un organigramme. Il nous reste encore à apprendre comment représenter une boucle FOR ... NEXT dans un organigramme. Vous trouverez à la page suivante l'organigramme du dernier programme d'exemple du chapitre sur les boucles FOR ... TO ... NEXT.

ORGANIGRAMME DU PROGRAMME FOR ... TO ... NEXT



Vous connaissez déjà tous les symboles utilisés dans cet organigramme. Le premier rectangle affecte la valeur 1 comme valeur de départ au compteur A. Ensuite vient la sortie d'une sous-chaîne avec LEFT$(A$,A). Le compteur est alors incrémenté de 1. Le losange demande si le compteur est déjà plus grand que le nombre de caractères qui composent A$. Tant que ce n'est pas le cas, le programme saute à travers le connecteur A au début de la boucle où se trouve l'autre connecteur A. Voici donc comment on représente une boucle dans un organigramme.

Si le compteur est plus grand que LEN(A$), c'est la seconde boucle qui entre en action. La seconde boucle a été représentée de la même manière que la première, si ce n'est que nous avons changé le nom des connecteurs pour les distinguer de ceux de la première boucle.

Vous disposez maintenant de tout l'arsenal de symboles nécessaire pour réaliser des organigrammes pour tout type de programme. Nous en aurons bientôt fini avec l'étude des structures de programmation lorsque nous aurons appris à manier les instructions de sauts calculés. C'est l'objet du paragraphe suivant.

3.3 INSTRUCTIONS DE SAUTS CALCULES

Les instructions de sauts calculés permettent d'arriver à un déroulement plus souple du programme alors que nous n'avons vu jusqu'ici que des instructions de saut à une ligne déterminée. Le numéro de ligne d'une instruction GOTO par exemple ne peut pas être influencé d'une façon ou d'une autre: avec GOTO 100, le programme sautera toujours à l'adresse 100. Il serait pourtant très intéressant de pouvoir entrer une valeur au début d'un programme, en fonction de laquelle le programme sauterait à une ligne différente. Ceci peut d'ailleurs être réalisé avec l'instruction IF ... THEN mais que de façon trés lourde comme le montre l'exemple suivant :

```
10 REM SAUT A DIFFERENTES LIGNES
20 PRINT"Entrez un nombre entre";
30 PRINT "1 et 4."
40 PRINT
50 INPUT"Quel nombre";Z
60 IF Z = 1 THEN 100
70 IF Z = 2 THEN 200
80 IF Z = 3 THEN 300
90 IF Z = 4 THEN 400
100 PRINT"Saut en ligne 100"
110 GOTO 410
200 PRINT"Saut en ligne 200"
210 GOTO 410
300 PRINT"Saut en ligne 300"
310 GOTO 410
400 PRINT"Saut en ligne 400"
410 END
```

Ce programme arrive bien au résultat recherché : suivant le nombre entre 1 et 4 que vous entrez, le programme saute à une ligne différente. Mais vous voyez aussi que nous sommes obligé d'utiliser une ligne différente avec l'instruction IF ... THEN pour chacune des possibilités.

L'instruction ON ... GOTO va nous permettre d'obtenir ce même résultat de façon plus simple et plus rapide :

ON (variable) **GOTO** (numéro de ligne),(numéro de ligne ou label)

Cette extension de l'instruction GOTO permet en effet de réaliser des sauts calculés à différentes lignes de programme, suivant la valeur de la variable placée à la suite de ON. La valeur de cette variable doit être comprise entre 0 et le nombre de numéros de ligne placés à la suite de GOTO. Si cette variable a une valeur non entière, les décimales sont ignorées. Si cette valeur est négative, elle sera ignorée.

Si la variable est nulle ou si elle a une valeur supérieure au nombre de numéros de ligne placés après GOTO, c'est l'instruction qui suit l'instruction ON ... GOTO qui sera exécutée. Voici quelques exemples simples pour bien comprendre le mécanisme de cette instruction :

EXEMPLES :

a)
```
10 ON Z GOTO 100,200,250,300
20 PRINT
.
.
.
```

Si la variable Z vaut 1, le programme saute en ligne 100. Si la valeur Z acquiert par la suite (dans une boucle par exemple) les valeurs 2, 3 puis 4, le programme sautera respectivement en ligne 200, 250 puis 300. Si Z vaut ensuite 5 ou plus, le programme passera à l'exécution de l'instruction suivante, le PRINT qui figure en ligne 20.

b)
```
10 ON Z+3/4 GOTO 100,200,300
20 PRINT
.
.
.
```

Vous voyez que vous pouvez aussi utiliser une expression arithmétique à la place d'une variable. L'intérêt principal de l'instruction ON ... GOTO est qu'elle remplace plusieurs instructions IF ... THEN, ce qui permet au programmeur de gagner du temps et également d'économiser de la place en mémoire BASIC. Voici par exemple une version améliorée de l'exemple précédent, cette fois avec l'instruction ON ... GOTO :

```
10 REMSAUT A DIFFERENTES LIGNES AVEC ON ... GOTO
20 PRINT"Entrez un nombre entre";
30 PRINT  "1 et 4."
40 PRINT
50 INPUT"Quel nombre";Z
60 ON Z GOTO 100,200,300,400
100 PRINT"Saut en ligne 100"
110 GOTO 410
200 PRINT"Saut en ligne 200"
210 GOTO 410
300 PRINT"Saut en ligne 300"
310 GOTO 410
400 PRINT"Saut en ligne 400"
410 END
```

Vous voyez que nous avons économisé trois lignes de programme. Sur des programmes plus longs, la place ainsi gagnée peut être considérable.

Nous avons d'autre part utilisé dans cet exemple une technique qui peut se révèler très intéressante lorsqu'il s'agit de développer des programmes d'une taille relativement importante. Nous avons en effet volontairement numéroté les différentes lignes auxquelles le programme doit sauter de 100 en 100 alors que ce n'était pas nécessaire a priori. Lorsqu'on réalise l'organigramme d'un tel programme, les différentes lignes sont représentées par des branchements horizontaux. Mais comme on ne peut pas savoir avant d'avoir développé le programme quels numéros de ligne correspondront à ces divers branchements, on choisit volontairement des numéros de ligne avec un écart suffisamment important pour que les différentes parties du programme puissent entrer entre ces numéros de ligne.

Les numéros de ligne figurant dans une instruction ON ... GOTO représentent en effet souvent différentes parties d'un programme dont chacune a une fonction spécifique. Il est donc intéressant de faire commencer chacune de ces parties par un numéro "rond".

On peut donc espacer ces numéros de ligne de 100 en 100 comme dans notre exemple ou de 1000 en 1000, etc... Ceci permet de rendre le listing du programme plus lisible.

3.3.1 PROGRAMME D'EXEMPLE "COURS DE CALCUL"

Nous connaissons maintenant un jeu d'instructions suffisamment important pour que nous puissions oser nous attaquer à un projet relativement ambitieux. Supposons donc que vous vouliez réaliser pour vos enfants un cours de calcul avec les quatre opérateurs arithmétiques de base sur l'Atari ST. Ce cours devra avoir les caractéristiques suivantes :

1. Choix d'une des quatres opérations fondamentales ou fin du programme
2. Création d'un exercice qui devra être trouvé en un maximum de trois tentatives
3. Après l'échec de la troisième tentative, le résultat devra être affiché
4. Entrée du plus grand nombre pouvant entrer en jeu dans chaque exercice
5. Demander après chaque exercice si l'utilisateur veut faire un autre exercice avec le même type d'opérations

Voici maintenant le listing de notre programme de cours de calcul. Comme ce programme est tout de même assez long, voici comment vous pourrez le sauvegarder sur disquette après l'avoir entré: placez d'abord une disquette dans votre lecteur de disquette et, après avoir entré le programme dans l'ordinateur, entrez l'instruction suivante :

```
SAVE CALCUL
```

Le programme sera alors sauvegardé automatiquement sur la disquette.

Si vous voulez réutiliser plus tard votre programme, vous n'aurez plus qu'à le recharger de la disquette dans l'ordinateur. Utilisez pour cela les instructions LOAD ou OLD.

LISTING DU PROGRAMME 'COURS DE CALCUL'

```
5    REM ***** MENU *****
10   FULLW 2:CLEARW 2:F=0
20   PRINT
30   PRINT TAB(12)"COURS DE CALCUL"
40   PRINT:PRINT
50   PRINT TAB(12)"Choisissez:"
60   PRINT
70   PRINT TAB(12)"1 pour l'addition"
80   PRINT
90   PRINT TAB(12)"2 pour la soustraction"
100  PRINT
110  PRINT TAB(12)"3 pour la division"
120  PRINT
130  PRINT TAB(12)"4 pour la multiplication"
133  PRINT
135  PRINT TAB(12)"5 pour FIN"
140  PRINT
150  PRINT TAB(12);:INPUT"Quel nombre";Z
160  IF Z<1 OR Z>5 THEN 10
170  ON Z GOTO 200,600,1000,1300,1600
200  REM **********
210  REM  ADDITION
220  REM **********
230  CLEARW 2
240  PRINT TAB(10)"Entrez le plus grand nombre"
250  PRINT
260  PRINT TAB(10)"pour l'addition."
270  PRINT
290  PRINT TAB(10);:INPUT"Plus grand nombre";GR
```

```
299  REM
300  REM GENERATION DE NOMBRES ALEATOIRES
301  REM
310  A1=INT(GR*RND(1))+1
320  A2=INT(GR*RND(1))+1
329  REM
330  REM CALCUL DU RESULTAT
331  REM
340  ER=A1+A2
350  CLEARW 2
360  PRINT
370  PRINT A1 "+" A2 "=";
380  INPUT ES
390  IFES=ERTHENPRINT:PRINTTAB(10)"C'ESTJUSTE!":F=0:GOTO450
400  PRINT:PRINT TAB(10)"C'EST FAUX!"
410  FOR I=0 TO 2000:NEXT I
420  F=F+1
430  IF F<=2 THEN 350
440  PRINT
450  FOR I=0 TO 2000:NEXT I
460  PRINT TAB(5) "Le résultat est";ER
470  FOR I=0 TO 3000:NEXT I
480  PRINT TAB(5) "Encore un exercice O/N";
490  INPUT A$
500  IF A$="O" THEN F=0:GOTO 300
510  GOTO 10
600  REM **************
610  REM  SOUSTRACTION
620  REM **************
630  CLEARW 2
640  PRINT TAB(10)"Entrez le plus grand nombre"
650  PRINT
660  PRINT TAB(10)"pour la soustraction."
670  PRINT
690  PRINT TAB(10);:INPUT"Plus grand nombre";GR
699  REM
700  REM GENERATION DE NOMBRES ALEATOIRES
```

```
701  REM
710  A1=INT(GR*RND(1))+1
720  A2=INT(GR*RND(1))+1
729  REM
730  REM CALCUL DU RESULTAT
731  REM
740  IF A1 < A2 THEN I=A1:A1=A2:A2=I
750  CLEARW 2
760  PRINT
770  ER=A1-A2
780  PRINT A1 "-" A2 "=";
790  INPUT ES
800  IFES=ERTHENPRINT:PRINTTAB(10)"C'ESTJUSTE!":F=0:GOTO860
810  PRINT:PRINT TAB(10)"C'EST FAUX!"
820  FOR I=0 TO 2000:NEXT I
830  F=F+1
840  IF F<=2 THEN 750
850  PRINT
860  FOR I=0 TO 2000:NEXT I
870  PRINT TAB(5) "Le résultat est";ER
880  FOR I=0 TO 3000:NEXT I
890  PRINT TAB(5) "Encore un exercice O/N";
900  INPUT A$
910  IF A$="O" THEN F=0:GOTO 710
920  GOTO 10
1000 REM **********
1001 REM  DIVISION
1002 REM **********
1010 CLEARW 2
1020 PRINT TAB(10)"Entrez le plus grand nombre"
1030 PRINT
1040 PRINT TAB(10)"pour la division."
1050 PRINT
1070 PRINT TAB(10);:INPUT"Plus grand nombre";GR
1079 REM
1080 REM GENERATION DE NOMBRES ALEATOIRES
1081 REM
```

```
1090 A1=INT(GR*RND(1))+1
1100 A2=INT(GR*RND(1))+1
1109 REM
1110 REM CALCUL DU RESULTAT
1111 REM
1120 ER=A1*A2
1130 CLEARW 2
1140 PRINT
1150 PRINT ER "/" A1 "=";
1160 INPUT ES
1170 IFES=A2THENPRINT:PRINTTAB(10)"C'ESTJUSTE!":F=0:GOTO1240
1180 PRINT:PRINT TAB(10)"C'EST FAUX!"
1190 FOR I=0 TO 2000:NEXT I
1200 F=F+1
1210 IF F<=2 THEN 1130
1220 PRINT
1230 FOR I=0 TO 2000:NEXT I
1240 PRINT TAB(5) "Le résultat est";A2
1250 FOR I=0 TO 3000:NEXT I
1260 PRINT TAB(5) "Encore un exercice O/N";
1270 INPUT A$
1280 IF A$="O" THEN F=0:GOTO 1090
1290 GOTO 10
1300 REM **********
1301 REM  MULTIPLICATION
1302 REM **********
1310 CLEARW 2
1320 PRINT TAB(10)"Entrez le plus grand nombre"
1330 PRINT
1340 PRINT TAB(10)"pour la multiplication."
1350 PRINT
1370 PRINT TAB(10);:INPUT"Plus grand nombre";GR
1379 REM
1380 REM GENERATION DE NOMBRES ALEATOIRES
1381 REM
1390 A1=INT(GR*RND(1))+1
1400 A2=INT(GR*RND(1))+1
```

```
1409 REM
1410 REM CALCUL DU RESULTAT
1411 REM
1420 ER=A1*A2
1430 CLEARW 2
1440 PRINT
1450 PRINT A1 "*" A2 "=";
1460 INPUT ES
1470 IFES=ERTHENPRINT:PRINTTAB(10)"C'ESTJUSTE!":F=0:GOTO1540
1480 PRINT:PRINT TAB(10)"C'EST FAUX!"
1490 FOR I=0 TO 2000:NEXT I
1500 F=F+1
1510 IF F<=2 THEN 1420
1520 PRINT
1530 FOR I=0 TO 2000:NEXT I
1540 PRINT TAB(5) "Le résultat est";ER
1550 FOR I=0 TO 3000:NEXT I
1560 PRINT TAB(5) "Encore un exercice O/N";
1570 INPUT A$
1580 IF A$="O" THEN F=0:GOTO 1390
1590 GOTO 10
1600 CLEARW 2
1610 END
```

Examinons maintenant les points les plus importants de ce listing. Les lignes 10 à 150 construisent ce qu'on appelle un menu. Un menu est une partie du programme où l'utilisateur peut choisir entre diverses fonctions du programme. Tout programme qui se respecte doit proposer au moins un menu.

La ligne 150 vous demande d'entrer un nombre correspondant au point du programme que vous choisissez. La ligne 160 teste si le nombre que vous avez entré est valable. C'est un bel exemple d'opération logique: si un nombre entré est SOIT inférieur à 1 SOIT supérieur à 5, le programme saute à la ligne 10. Il suffit qu'une seule de ces conditions soit réalisée pour que le branchement se produise, c'est pourquoi nous avons utilisé l'opérateur logique OR (ou inclusif).

La ligne 170 nous montre ensuite un exemple d'emploi de l'instruction ON ... GOTO. Si Z vaut 1, on saute à la ligne 200.
Si Z vaut 2, on saute à la ligne 600, etc... Les différentes opérations de calcul sont donc appelées par l'entrée d'un nombre. Nous avons ici économisé 5 lignes de IF ... THEN.

Les lignes 200 à 220 servent à souligner le début de la partie du programme qui traite l'addition. Une telle présentation est très appréciable pour le programmeur quand il a affaire à un programme assez long. Elle permet en effet de se repérer plus facilement dans le listing et donc de retrouver plus vite une ligne déterminée du programme.

La ligne 230 vide l'écran. Les lignes suivantes demandent à l'utilisateur d'entrer un nombre qui sera la valeur maximum que pourront atteindre les nombres à additionner. Deux nombres aléatoires A1 et A2 sont ensuite produits, à partir desquels sera créé l'exercice d'addition. La ligne 370 présente l'exercice sur l'écran. La ligne 380 permet à l'utilisateur d'entrer un résultat que la ligne 390 compare alors à la valeur calculée en ligne 340. Si le résultat entré est faux, la ligne 400 sort d'abord une ligne d'espaces sur l'écran puis le message "C'est faux!". La ligne 410 est une boucle de temporisation permettant à l'utilisateur de bien lire le message.

La ligne 420 incrémente un compteur qui compte le nombre de réponses fausses. Nous voulions en effet que le résultat correct soit affiché au bout de trois réponses fausses. La ligne 430 teste si le nombre de trois réponses fausses est déjà atteint. Tant que ce n'est pas le cas, le programme retourne en ligne 350 pour présenter l'exercice à nouveau. Si trois réponses fausses ont été données, le programme passe à l'exécution de la suite du programme, en ligne 440. Si une réponse juste a été donnée, le programme saute de la ligne 390 à la ligne 450 qui est une boucle de temporisation. La ligne 460 affiche ensuite le résultat. Après une autre temporisation, le programme demande en ligne 480 si l'utilisateur veut faire un autre exercice.

Si l'utilisateur entre ici un "O", le compteur F est d'abord remis à 0 (réinitialisé) et le programme saute en ligne 300 où le programme tire deux nouveaux nombres au sort et propose un nouvel exercice.

Si l'utilisateur entre autre chose que "O", le programme retourne en ligne 10 où le menu est à nouveau présenté.

Le compteur F est remis à 0 (réinitialisé) avant tout nouvel exercice car sinon le test IF F<=2 en ligne 430 serait négatif dès la première réponse fausse du second exercice. F vaudrait en effet déjà 3. Il convient donc, lorsqu'on utilise plusieurs fois dans un programme de tels compteurs, de ne jamais oublier de les réinitialiser chaque fois que nécessaire.

Les autres parties du programme, "soustraction", "division" et "multiplication" sont construites fondamentalement de la même façon et ne se distinguent que par les petites adaptations qui ont été rendues nécessaires pour chaque opération. Voici par exemple comment a été réalisée la partie du programme pour la soustraction.

La ligne 740 permet d'empêcher des résultats négatifs. En effet, si A1 est supérieur à A2, l'exercice pourra être proposé sans problème. Si par contre A2 est supérieur à A1, A1 - A2 donnera un résultat négatif, ce que nous voulons éviter. C'est pourquoi nous avons ajouté la ligne 740 qui échange dans ce cas les valeurs des deux variables A1 et A2. Notez à cet égard qu'un échange ne peut être programmé ainsi :

A1=A2:A2=A1 **< I N C O R R E C T ! >**

En effet dans ce cas A1 se verrait d'abord affecter la valeur de A2. A2 se verrait ensuite affecter la nouvelle valeur de A1, c'est-à-dire la valeur de A2. A2 n'aurait donc pas changé de valeur, et A1 serait égal à A2.

Pour échanger entre elles les valeurs de deux variables, il faut donc absolument "sauver" la valeur d'une de ces variables, c'est-à-dire l'affecter à une variable provisoire utilisée uniquement pour l'échange. Ici, la valeur de A1 est sauvée dans la variable I. La valeur de A2 est affectée à A1 et l'ancienne valeur de A1, c'est-à-dire la valeur de I, est affectée à A2. Cette technique d'échange des variables est très importante et vous aurez certainement souvent à y recourir. Assurez-vous donc que vous l'avez bien comprise.

Ce point était la seul différence importante avec l'addition. Nous avons également utilisé pour la division un petit "truc" qui est propre à cette partie du programme. Le but était toujours le même, à savoir s'assurer que le diviseur ne serait pas plus grand que le dividende et que le résultat serait un nombre entier. Nous avons donc en ligne 1120 calculé le résultat en multipliant A1 par A2, comme pour la multiplication. Mais pour l'exercice, c'est le "résultat" ER qui est divisé par A1. Le résultat à trouver est donc A2 qui est toujours un nombre entier.

Il n'y a pas de remarque particulière à faire sur la multiplication.

Voilà ce que nous avions à dire sur ce programme qui vous montre un exemple, très proche de la programmation pratique, d'une utilisation de l'instruction ON ... GOTO. Avant que nous ne vous proposions de tester vos nouvelles connaissances, nous allons aborder une instruction très voisine de l'instruction ON ... GOTO, l'instruction ON ERROR GOTO. Mais résumons auparavant les principaux points qu'il faut retenir pour utiliser ON ... GOTO :

1. *La valeur qui suit le mot ON peut être une variable, une constante ou une expression arithmétique. Elle désigne la place du numéro de la ligne à laquelle le programme doit sauter dans la liste de numéros de ligne qui suit le mot GOTO.*

2. *Si cette valeur est nulle ou si elle est plus grande que le nombre de numéros de ligne suivant GOTO, c'est l'instruction suivant l'instruction ON ... GOTO qui sera exécutée.*

3. *ON ... GOTO remplace plusieurs instructions IF ... THEN.*

3.3.2 SAUTS AVEC ON ERROR GOTO

Cette variante particulière de l'instruction ON ... GOTO est utilisée pour faire gérer des erreurs qui pourraient se produire par le programme lui-même. L'instruction ON ERROR GOTO est placée le plus souvent au début du programme et l'ordinateur "note" dans ses tablettes à quel endroit du programme cette instruction est apparue pour la première fois. Toutes les erreurs qui se produisent après cette instruction font que le programme saute à la ligne dont le numéro suit l'instruction ON ERROR GOTO. Ce numéro doit correspondre à un petit programme qui réagisse en fonction de l'erreur. Mais pour réaliser ce programme, encore faut-il savoir quelle erreur s'est produite, et en quelle ligne du programme.

L'Atari ST dispose de deux variables qui peuvent être interrogées à ce sujet :

ERR

et

ERL

Si vous entrez ERR, vous obtenez le numéro de l'erreur qui s'est produite. Vous pouvez alors voir dans l'annexe 8 de votre manuel à quelle erreur correspond ce numéro d'erreur. Vous pouvez donc faire sortir par votre petit programme de traitement des erreurs vos propres messages d'erreur, en fonction de la valeur de ERR et éviter ainsi une interruption inopinée du programme.

La variable ERL vous permet ensuite de demander le numéro de la ligne où l'erreur s'est produite.

Cette méthode de traitement des erreurs vous permet de faire exécuter dans le cours du programme un certain nombre de tâches avant que le programme ne soit interrompu définitivement.

Vous pouvez ainsi dans un programme de gestion de fichier par exemple fermer tous les fichiers dès qu'une erreur apparaît, ce qui évitera toute perte de données. Le programme peut alors sortir son propre message d'erreur avant d'être interrompu.

Il est même parfois possible de laisser le programme suivre son cours après l'apparition d'une erreur. Il suffit dans ce cas de sortir un message d'erreur pour l'utilisateur avant que le programme ne continue son cours normal. Il faut pour ce faire utiliser l'instruction :

RESUME (numéro de ligne)

qui indique à partir de quelle ligne le programme doit se continuer après une erreur.

Vous avez également la possibilité de faire sortir, dans un programme, le message d'erreur original de l'Atari ST. Vous devez utiliser pour cela la séquence d'instructions suivante :

ON ERROR GOTO 0

Malheureusement, l'instruction ON ERROR GOTO ne fonctionne pas parfaitement pour le moment. C'est ce que vous pourrez d'ailleurs constater avec le programme suivant :

```
10 ON ERROR GOTO 100
20 GOTO 40
30 FOR I=1 TO 20
40 PRINT I
```

```
50 NEXT
60 END
100 PRINT ERL,ERR
110 RESUME NEXT
```

Malgré la ligne 10, le programme s'interrompt en sortant le message d'erreur original. Toutefois, le numéro d'erreur et la ligne fautive ont bien été affectés aux deux variables. Il ne nous reste qu'à espérer que ce défaut sera bientôt corrigé.

Voici à nouveau des exercices suivis de leurs solutions. Nous vous invitons à toujours suivre les cinq étapes de programmation que nous avons décrites au début de cet ouvrage. Bonne chance !

EXERCICES

1. Ecrivez un programme qui additionne les nombres de la "série harmonique" (1 + 1/2 + 1/3 + 1/4 + 1/5 + ... + 1/n) jusqu'à obtenir une valeur entrée par l'utilisateur du programme. Le nombre d'additions effectuées sera affiché toutes les 50 additions. Il faudra indiquer en fin de programme le nombre total des valeurs qui ont été additionnées.

2. Ecrivez un programme qui calcule les points zéro réels d'une équation du second degré de forme :

$$AX^2 + BX + C = 0.$$

La formule est :

$$x1 = (-B + SQR(B^2-4AC)) / 2A$$
$$x2 = (-B - SQR(B^2-4AC)) / 2A$$

Notez qu'il n'y a pas de solutions réelles pour $B^2-4AC < 0$.
Il faut en tenir compte dans votre programme.

4. Que se passera-t-il dans le programme ci-dessous si vous entrez 4 ? Essayez bien sûr de résoudre ce problème sans entrer le programme dans l'ordinateur.

```
10 REM EXERCICE TEST AVEC ON ... GOTO
20 INPUT"Entrez un nombre.";Z
30 ON Z GOTO 100,150,400:CLEARW 2
40 PRINT"Valeur non-valable."
50 END
100 PRINT"LIGNE 100"
110 END
150 PRINT"LIGNE 150"
.
.
```

3.4 L'INTERROGATION DU CLAVIER

Le BASIC de l'Atari ST vous offre plusieurs possibilités pour interroger le clavier. La possibilité la plus simple, et que vous connaissez déjà bien, consiste à utiliser l'instruction INPUT pour lire des caractères et les affecter à une variable.

Il existe en outre également la fonction INKEY$ mais, malheureusement, dans la version BASIC actuelle, elle ne fonctionne pas correctement. Si vous entrez par exemple la ligne de programme suivante :

```
10 A$=INKEY$:IF A$="" THEN 10
```

et que vous lancez ensuite ce petit programme, vous constaterez que le programme ne s'arrête pas, même si vous actionnez une touche. Vous ne pouvez sortir de cette fonction que par l'emploi 'brutal' de **CONTROL C**.

Une autre possibilité de sélection des touches consiste à employer INPUT$:

```
10 A$=INPUT$(1)
20 A=ASC(A$)
30 PRINT A
```

Le paramètre entre parenthèses détermine combien de caractères vous voulez entrer. Dans notre exemple, ce sera donc un caractère qui sera entré et dont le code ASCII sera déterminé. L'avantage de cette fonction est que vous n'êtes pas obligé, après avoir entré un caractère, d'appuyer sur la touche RETURN pour que votre entrée soit prise en compte.

Toutes les fonctions que nous avons décrites jusqu'ici ont toutefois un inconvénient : elles ne permettent pas d'interroger toutes les touches du clavier.

C'est ainsi que les touches curseur et les touches de fonction ne transmettront aucune valeur si vous utilisez les fonctions dont nous avons parlé.

La solution nous est fournie ici par la fonction :

INP(2)

Le paramètre 2 désigne le clavier. Les paramètres autorisés pour INP sont :

0 = imprimante
1 = RS-232
2 = console (clavier et écran)
3 = interface MIDI
4 = processeur clavier

Le petit programme suivant vous montre les codes de toutes les touches. Vous pouvez l'interrompre avec la touche ESC.

```
10 A=INP(2)
20 PRINT A,CHR$(A)
30 IF A=27 THEN END
40 GOTO 10
```

Nous en avons fini avec la description des principales fonctions d'entrée de caractères à travers le clavier. Examinons maintenant quelques instructions ou fonctions qui ne sont pas utilisées très souvent dans les programmes. Il serait cependant bon que vous sachiez à "qui" vous avez affaire si vous rencontrez un jour une de ces instructions dans un programme.

3.5 FRE, POS, CALL, WAIT

Comme nous venons de l'indiquer, ces instructions sont assez rarement utilisées dans des programmes BASIC.

Cela n'a d'ailleurs rien à voir avec leur importance et leur utilité. Examinons-les donc dans l'ordre.

FRE(X)

indique le nombre d'octets disponibles en mémoire BASIC. En mode direct, vous pouvez entrer :

```
PRINT FRE(X)
```

qui donnera le nombre d'octets encore disponibles pour le BASIC.

POS

fournit la position actuelle du curseur sur la ligne de l'écran sur laquelle il se trouve. Entrez par exemple en mode direct :

```
PRINT "TEST" POS(X);"TEST A" POS(X)
```

et appuyez sur la touche RETURN. Vous obtenez :

```
TEST 4 TEST A 13
```

Le 4 correspond à la colonne où se trouvait le curseur après exécution de la première instruction PRINT, le 13 à celle où il était après la seconde instruction PRINT. POS vous permet donc de déterminer dans quelle position de la ligne serait effectuée la prochaine instruction PRINT.

CALL

vous permet d'appeler un programme en langage-machine commençant à l'adresse X. Vous pouvez ainsi appeler vos propres programmes en langage-machine ou des routines du système d'exploitation.

Vous n'aurez certainement pas à vous servir au début de cette instruction qui n'a d'intérêt que si vous maîtrisez le langage-machine ou si vous connaissez bien le système d'exploitation de l'Atari ST. Par ailleurs, si vous l'utilisez de manière inconsidérée, vous pouvez entraîner un "plantage" de la machine. Prudence donc !

WAIT

Avec l'instruction WAIT, vous pouvez bloquer le déroulement d'un programme jusqu'à ce qu'une CASE MEMOIRE ait pris une VALEUR DETERMINEE. Plus précisément, avec WAIT le programme attend un modèle bits déterminé dans une case mémoire. Cette instruction est, elle aussi, assez rarement utilisée.

Voilà en ce qui concerne les instructions FRE, POS, CALL et WAIT.

3.6 READ, DATA ET RESTORE

Nous n'avons jusqu'ici appris à entrer des données qu'à partir du clavier. Mais si notre programme doit traiter un grand nombre de données, qu'il s'agisse de variables numériques ou de chaînes de caractères, il est bien sûr trés pénible de devoir entrer ces données chaque fois qu'on relance le programme. Les instructions **READ** et **DATA** permettent de résoudre ce problème.

L'instruction **DATA** se compose de listes de données numériques ou alphanumériques (chaînes de caractéres) séparées entre elles par des virgules.

READ vous permet de lire les données dans l'ordre où elles figurent dans les instructions DATA. Il faut cependant faire attention à ce que le type de variable suivant l'instruction READ corresponde bien au type de donnée à lire. Vous ne pouvez pas faire lire une chaîne de caractères placée en DATA par une variable numérique.

Les lignes de DATA peuvent figurer en n'importe quel endroit du programme, au début, à la fin ou au milieu. Si le programme rencontre une instruction READ, il va rechercher automatiquement la première donnée placée dans une instruction DATA qu'il n'a pas encore lue. Il affecte alors cette donnée à la variable suivant l'instruction READ. Par exemple :

```
10 READ X
20 PRINT X
30 DATA 50
40 END
```

Vous obtenez 50. La ligne 10 affecte en effet à la variable X la valeur 50 lue par l'instruction READ. La ligne 20 sort alors la valeur de X et la ligne 30 n'a plus aucune influence sur le déroulement ultérieur du programme. Modifiez par exemple votre programme ainsi :

```
10 READ X,Y,Z
20 PRINT X,Y,Z
30 DATA 10,20,30
40 END
```

Après avoir lancé le programme, vous obtenez à l'écran :

```
10          20          30
```

READ a donc lu d'abord la première donnée figurant en DATA (10) qu'il a affectée à la variable X. READ a ensuite lu la seconde donnée figurant en DATA (20) qu'il a affectée à la variable Y. READ a lu enfin la troisième donnée figurant en DATA (30) qu'il a affectée à la variable Z. L'ordinateur lit donc chaque fois la donnée suivante grâce à ce qu'on appelle un pointeur, qui est dirigé sur la prochaine donnée à lire après chaque instruction READ. Au début du programme, ce pointeur est donc dirigé sur le premier élément de la première ligne de DATA.

En voici une illustration (le pointeur sera représenté par une flèche) :

```
30  DATA 10, 20, 30
       ^
```

Après l'exécution de la première instruction READ du programme, le pointeur est augmenté :

```
30  DATA 10, 20, 30
           ^
```

Après l'exécution de la seconde instruction READ du programme, le pointeur est à nouveau augmenté. Mais lorsque l'ordinateur arrive à la fin d'une liste de données, le pointeur n'est pas automatiquement dirigé à nouveau sur le premier élément mais il reste au contraire dirigé sur ce qui suit le dernier élément. Si on tente alors à nouveau de lire une donnée de la liste avec READ, l'ordinateur sort le message :

READ statement ran out of data at line (numéro de ligne)

(l'instruction READ a épuisé les données disponibles). Si vous voulez utiliser plusieurs fois une même liste, il faut utiliser l'instruction :

RESTORE

qui dirige à nouveau le pointeur sur le premier élément de la liste. Mais entrez d'abord le programme suivant pour voir ce qui se produit lorsque le programme essaye de lire plus de données que la liste n'en comporte :

```
10 READ A,B,C
20 PRINT A,B,C
30 DATA 10,20,30
40 READ D,E,F
```

```
50 PRINT D,E,F
60 END
```

Après que les valeurs 10, 20 et 30 aient été sorties, le message d'erreur :

READ statement ran out of data at line 40

apparaît. La ligne 40 a en effet essayé de lire un quatrième élément de DATA alors qu'il n'y en avait que 3. Vous pouvez remédier à cette erreur soit en ajoutant trois données supplémentaires en DATA, soit en utilisant RESTORE pour réinitialiser le pointeur de données :

```
10 READ A,B,C
20 PRINT A,B,C
30 DATA 10,20,30
35 RESTORE
40 READ D,E,F
50 PRINT D,E,F
60 END
```

Dans cette nouvelle version (vous n'avez à entrer que la ligne 35) le programme fonctionne sans erreur et sort :

```
10          20          30
10          20          30
```

Le pointeur de données a en effet été réinitialisé et les variables D, E et F se sont également vu affecter les valeurs 10, 20 et 30.
Notez qu'une seule instruction permet de lire plusieurs données et de les affecter à plusieurs variables différentes, comme nous l'avons fait avec READ A,B,C et READ D,E,F. Vous pouvez également affecter les différentes valeurs à lire successivement à la même variable :

```
10 FOR I=1 TO 3
20 READ X
30 PRINT X
40 NEXT I
50 DATA 10,20,30
60 END
```

Nous avons ici placé les instructions READ X et PRINT X dans une boucle qui sera parcourue trois fois. Lors de chaque parcours, c'est la prochaine valeur figurant en DATA qui sera affectée à l'instruction X puis sortie avec PRINT X.

Comme nous l'avons déjà indiqué, vous pouvez également placer des chaînes de caractères en DATA. Celles-ci n'ont pas en principe à être mises entre guillemets. Mais il y a comme toujours des exceptions à respecter pour éviter des déboires :

La virgule, le double-point et le point-virgule doivent être entre guillemets. Mais pensez surtout à ne pas lire avec une variable numérique une chaîne de caractères.

Si vous avez créé une longue liste de données mixtes, une telle erreur peut se produire facilement :

```
10 FOR I=1 TO 3
20 READ A,B,C$
30 PRINT A,B,C$
40 NEXT I
50 DATA 10,20,TEST 1,30,40,TEST 2,50,TEST 3,OK
60 END
```

Jusqu'au deuxième parcours de la boucle FOR ... NEXT, notre programme tournera sans problème car jusque là les types de variables correspondent aux données effectivement placées en mémoire : l'instruction READ prévoit que deux variables numériques doivent être lues puis une variable alphanumérique et ainsi de suite. Les données placées en ligne 50 correspondent à cet

ordre jusqu'à la septième donnée, 50. Mais ensuite, le programme va essayer d'affecter la chaîne de caractères TEST 3 à la variable numérique B, ce qui n'est pas possible, comme vous le savez et qui provoque donc une interruption du programme lors du troisième parcours de la boucle avec le message d'erreur :

Function call not allowed at line 20

Nous avons appris beaucoup de choses sur la transmission de données à l'ordinateur ou au programme. Nous pouvons entrer des données directement avec INPUT, INPUT$ ou INP ou les placer dans des lignes de DATA qui seront d'ailleurs sauvegardées avec le programme si vous le sauvegardez sur disquette ! Par contre, pour sauvegarder des données entrées avec INPUT il faut les écrire sur un fichier séparé sur disquette.

Il nous faut signaler une application très courante des instructions READ et DATA que vous n'utiliserez toutefois peut-être pas dans l'immédiat et qui consiste à générer des programmes en langage-machine à partir du BASIC. Il s'agit le plus souvent de nombreuses lignes de DATA qui sont lues par une boucle FOR ... NEXT et placées par l'instruction POKE dans les adresses correspondantes. Après quoi le programme en langage-machine se trouve en mémoire et peut être appelé avec l'instruction CALL.

En tout cas les instructions READ, DATA et RESTORE sont très importantes et nous vous invitons à relire attentivement ce chapitre si vous n'êtes pas sûr d'avoir tout assimilé parfaitement. Nous allons en effet aborder maintenant les tableaux et vous verrez que nous aurons de nombreuses occasions de recourir aux instructions READ et DATA.

CHAPITRE 4

APPLICATIONS BASIC PLUS COMPLEXES

4.1 LES TABLEAUX

La programmation des tableaux est certainement la technique la plus difficile à assimiler pour un débutant en BASIC. C'est pourquoi il est nécessaire d'aborder cette technique avec un esprit de méthode. Comme toujours, nous vous guiderons par des exemples très simples au départ puis de plus en plus complexes.

4.1.1 TABLEAUX A UNE DIMENSION

Supposez que vous vouliez écrire un programme calculant votre revenu mensuel moyen sur une période de 12 mois. Nous utiliserons tout d'abord une boucle pour entrer les 12 revenus mensuels :

```
5   CLEARW 2:FULLW 2
10  REM revenu mensuel moyen
20  REM calculé sur 12 mois
30  FOR I=1 TO 12
40  INPUT"Revenu mensuel";M
50  S=S+M
60  NEXT I
70  DU=S/12
80  DU=INT(DU*100)/100
90  PRINT"Revenu moyen";
100 PRINT DU;"F"
110 END
```

Après avoir lancé ce programme, entrez 12 valeurs.

Vous obtenez le revenu mensuel moyen arrondi à deux décimales. Une boucle FOR ... NEXT permet d'entrer les différentes valeurs des revenus mensuels et la ligne 50 calcule au fur et à mesure la somme de ces revenus. La ligne 70 calcule le revenu mensuel moyen et la ligne 80 l'arrondit à deux décimales.

Nous avons donc bien calculé le revenu mensuel moyen, mais nous ne pourrions plus obtenir du programme le détail des différents revenus mensuels, par exemple le revenu du mois de mai. Pour conserver les valeurs des différents revenus mensuels, il faudrait que nous n'utilisions pas une seule et même variable pour entrer ces revenus. Il suffirait donc d'utiliser 12 variables, une pour chaque mois :

```
5   CLEARW 2:FULLW 2
10  REM revenu mensuel moyen
20  REM sauver les differentes valeurs mensuelles
30  INPUT"Revenu mensuel 1";M1
40  INPUT"Revenu mensuel 2";M2
50  INPUT"Revenu mensuel 3";M3
60  INPUT"Revenu mensuel 4";M4
70  INPUT"Revenu mensuel 5";M5
80  INPUT"Revenu mensuel 6";M6
90  INPUT"Revenu mensuel 7";M7
100 INPUT"Revenu mensuel 8";M8
110 INPUT"Revenu mensuel 9";M9
120 INPUT"Revenu mensuel 10";M10
130 INPUT"Revenu mensuel 11";M11
140 INPUT"Revenu mensuel 12";M12
150 S=M1+M2+M3+M4+M5+M6+M7+M8+M9+M10+M11+M12
160 DU=S/12
170 DU=INT(DU*100)/100
180 PRINT"Revenu moyen";
190 PRINT DU;"F"
200 END
```

Si vous développez ce programme, vous pourrez réutiliser les valeurs pour les différents mois à tout moment, grâce aux 12 variables M1 à M12. Pour connaître la valeur du mois de mai, il vous suffira d'utiliser la variable M5.

Nous avons donc maintenant en permanence les 12 revenus mensuels à notre disposition, mais cela nous a tout de même coûté 9 lignes de programmation supplémentaires. D'autre part le maniement de 12 variables n'est pas très pratique: si vous voulez sortir plus tard les 12 valeurs correspondant aux revenus mensuels, vous devrez utiliser une instruction telle que celle-ci :

```
100 PRINT M1,M2,M3,M4,M5,M6,M7,M8,M9,M10,M11,M12
.
.
.
```

Evidemment le BASIC offre une solution plus élégante et plus pratique, les variables indicées ou tableaux. Vous pouvez ainsi créer des variables telles que celle-ci :

A(I)

où I est l'indice de la variable. Vous pouvez donc, pour rester dans notre exemple, faire entrer les 12 revenus avec une boucle qui fasse varier la valeur de I de 1 à 12 et vous pourrez ensuite demander la valeur de n'importe quel mois en utilisant l'indice de ce mois. La valeur du mois de mai par exemple nous sera donc fournie par A(5).

Nous avons comparé, au début de cet ouvrage, les variables à des tiroirs dans lesquels on peut placer, selon le type de variable, des valeurs numériques ou alphanumériques. On pourrait de même comparer un tableau à un placard à plusieurs tiroirs placés les uns au dessus des autres. Chaque tiroir porterait un numéro qui n'aurait rien à voir avec le contenu de ces tiroirs. Ce numéro est un **INDICE.**

L'indice est placé entre parenthèses à la suite de la variable indicée ou tableau :

A(1)

La valeur 1 est ici l'indice de la variable. Il ne faut pas confondre A(1) et A1! A1 est une variable ordinaire alors que A(1) est une des variables indicées du tableau A(X). Voici comment on pourrait représenter un tel tableau :

A(1)	2334
A(2)	2333
A(3)	2345.65
A(4)	2344.34
.	.
.	.
.	.
.	.
A(12)	3433.20

Vous voyez que le terme de tableau décrit bien la réalité. Notre tableau a ici 12 tiroirs dont chacun contient une valeur. Si nous voulons connaître la valeur du troisième élément, il nous suffit d'utiliser l'indice 3 :

PRINT A(3)

donnera donc :

2345.65

Mais si nous voulons utiliser de tels tableaux dans nos programmes, il nous faut indiquer tout d'abord à l'ordinateur les dimensions de notre tableau, c'est-à-dire combien d'éléments il devra contenir. Il nous faut pour cela employer l'instruction DIM dont la syntaxe est la suivante :

DIM nom du tableau (nombre d'éléments)

Dans notre exemple, nous devrions utiliser l'instruction :

DIM A(12)

où A est le nom du tableau et 12 le nombre maximum d'éléments ou de champs du tableau. L'instruction DIM est le plus souvent placée en début de programme. En effet, une fois qu'un tableau a été dimensionné, vous ne pouvez le redimensionner pendant tout le déroulement du programme. Sinon vous obtenez le message d'erreur :

? You defined an array more than once

Nous avons, dans l'exemple ci-dessus, dimensionné un tableau de variables à virgule flottante, c'est-à-dire de variables pouvant recevoir n'importe quels nombres réels. Voici des exemples de dimensionnement de tableaux de variables alphanumériques (chaînes de caractères), entières et à nouveau numériques avec virgule flottante :

```
DIM DE$(15)
DIM GZ%(20)
DIM AB(12)
```

Vous pouvez dimensionner plusieurs tableaux avec une seule instruction DIM :

```
DIM A(12),B$(16),S%(20)
```

Il convient de noter les deux particularités suivantes :

1. Si vous n'avez pas besoin de plus de 11 éléments ou champs dans votre tableau, il n'est pas nécessaire de le dimensionner. Il suffira en effet que vous appeliez un élément quelconque de ce tableau, par exemple A(4) pour que l'ordinateur exécute automatiquement l'instruction, en l'occurrence, DIM A(10).

2. L'indice de tout tableau commence à 0. La valeur utilisée pour dimensionner un tableau doit donc être inférieure de 1 au nombre d'éléments ou de champs que devra contenir votre tableau. Par exemple, DIM A(10) crée un tableau de 11 variables indicées, de A(0) à A(10).

Pour le moment nous ignorerons l'élément d'indice 0 du tableau, comme nous l'avons fait dans la figure précédente.

Si le dimensionnement d'un tableau de moins de 12 éléments ou champs est facultatif, il peut toutefois économiser de la place en mémoire si vous êtes sûr que votre tableau aura moins de 11 éléments, 6 par exemple. Dans ce cas vous pouvez utiliser l'instruction DIM X(6) par exemple.

Voyons maintenant ce que deviendrait notre programme de calcul du revenu mensuel moyen avec un tableau :

```
5   CLEARW 2:FULLW 2
10  REM revenu mensuel moyen
20  REM avec utilisation d'un tableau
30  DIM M(12)
40  FOR I=1 TO 12
50  INPUT"Revenu mensuel";M(I)
60  S=S+M(I)
70  NEXT I
80  DU=S/12
90  DU=INT(DU*100)/100
100 PRINT"Revenu moyen";
110 PRINT DU;"F"
120 END
```

Pour utiliser un tableau, nous n'avons eu besoin que d'une ligne d'instruction de plus que dans la première version du programme, pour introduire l'instruction DIM. Nous pouvons maintenant appeler à tout moment les variables correspondant aux différents revenus mensuels. Par exemple, si nous voulions sortir à nouveau un récapitulatif de tous les revenus mensuels avant de sortir le revenu mensuel, il suffirait de modifier ainsi la dernière partie du programme :

```
.
.
90  DU=INT(DU*100)/100
100 FOR I=1 TO 12
110 PRINT"REVENU MENSUEL"I,M(I)
120 NEXT I
130 PRINT"Revenu moyen";
140 PRINT DU;"F"
150 END
```

Nous avons donc ainsi trouvé avec les tableaux un moyen simple et pratique de garder disponibles pour une éventuelle utilisation ultérieure les revenus des différents mois.

Les tableaux qui s'écrivent ainsi :

A(X)

sont appelés tableaux unidimensionnels parce qu'ils ne possèdent qu'un indice.

L'indice peut être fourni aussi bien sous forme d'une variable ou d'une expression arithmétique que sous forme d'un nombre. Ceci vous permet de faire calculer l'indice en fonction de vos besoins. La même remarque s'applique également au nombre de champs ou éléments défini par l'instruction DIM. Vous pourriez donc parfaitement prévoir, toujours dans notre exemple de revenu mensuel moyen, que l'utilisateur puisse déterminer le nombre de mois dont il veut la moyenne :

```
.
.
.
30  INPUT"Combien de revenus mensuels";Z
40  DIM M(Z)
.
.
```

Le nombre de mois détermine ici les dimensions du tableau, ce qui permet d'adapter le programme aux besoins actuels de l'utilisateur et d'utiliser la mémoire disponible d'une façon optimale.

Si vous essayez d'appeler un élément d'un tableau qui sort du cadre fixé par l'instruction de dimensionnement DIM A(X), vous obtenez le message d'erreur :

Subscript refers to element outside the array

Par exemple, vous ne pouvez pas appeler l'élément A(16) si vous avez dimensionné le tableau A avec l'instruction DIM A(15).

Pour bien assimiler le fonctionnement des tableaux qui constituent une possibilité très importante de la programmation en BASIC, nous allons l'illustrer par quelques exemples avec des tableaux unidimensionnels. Nous utiliserons chaque fois les tableaux X et Y$ avec toujours 6 éléments. Nous continuerons pour le moment d'ignorer l'élément d'indice 0.

4.1.2 EXEMPLES D'APPLICATIONS DES TABLEAUX UNIDIMENSIONNELS

Lorsque vous venez de créer pour la première fois un tableau donné, tous ses éléments sont vides. Mais il se peut que vous ayez besoin d'annuler le contenu d'un tableau au cours du déroulement d'un programme. S'il s'agit d'un tableau numérique, vous pouvez annuler son contenu en affectant un 0 à tous les éléments du tableau :

```
10 REM EXEMPLE d'annulation TABLEAU NUMERIQUE
20 DIM X(6)
30 FOR I=1 TO 6
40 X(I)=0
50 NEXT I
60 END
```

Nous sommes parti dans cet exemple d'un tableau de 6 éléments (parce que nous oublions pour le moment le champ 0). La boucle FOR ... NEXT a reçu pour valeurs de départ et de fin 1 et 6, de façon à ce que le compteur I prenne successivement la valeur de tous les indices du tableau, de 1 à 6. La ligne 40 annule lors de chaque parcours l'élément d'indice I. Après le dernier parcours de la boucle, tous les éléments du tableau ont été ainsi annulés. Si nous n'avions pas utilisé une boucle, nous aurions dû écrire :

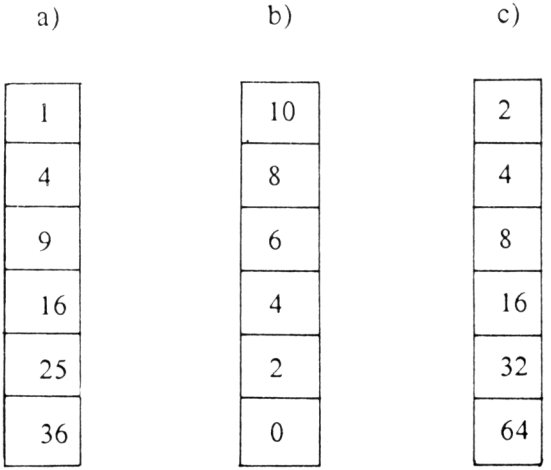
X(1)=0
X(2)=0
X(3)=0
X(4)=0
X(5)=0
X(6)=0

S'il s'agit maintenant d'annuler le contenu d'un tableau de variables alphanumériques (chaînes de caractères), vous ne pouvez bien sûr procéder de la même manière. Ce n'est pas le 0 (qui est aussi un caractère) ni même un espace mais une chaîne vide qu'il faut affecter à chaque élément du tableau :

```
10 REM exemple d'annulation tableau alphanumerique
20 DIM Y$(6)
30 FOR I=1 TO 6
40 Y$(I)=""
50 NEXT I
60 END
```

Rappelons que c'est une chaîne vide qu'il faut affecter à chaque élément du tableau. Il ne doit donc pas y avoir d'espace entre les deux guillemets à la ligne 40.

Venons-en maintenant à des exemples qui vous permettront de vous exercer au maniement des tableaux avec des boucles FOR ... NEXT. Soient 3 tableaux de 6 éléments chacun dont le contenu sera le suivant :



Comment pourrait-on créer ces trois tableaux avec une boucle FOR ... NEXT.

Exemple a)

Si vous réfléchissez un peu, vous constaterez que les contenus des divers éléments correspondent au carré de l'indice. Ce tableau aurait donc pu être créé avec un programme tel que celui-ci :

```
10 DIM X(6)
20 FOR I=1 TO 6
30 X(I)=I*I
40 NEXT I
50 END
```

Le fonctionnement de cette boucle est encore plus évident si on écrit sur le papier ce que le programme fait lors de chaque parcours de la boucle :

CREATION DU TABLEAU X(6)

I = 1	: X(1) = 1 * 1 = 1	1
I = 2	: X(2) = 2 * 2 = 4	4
I = 3	: X(3) = 3 * 3 = 9	9
I = 4	: X(4) = 4 * 4 = 16	16
I = 5	: X(5) = 5 * 5 = 25	25
I = 6	: X(6) = 6 * 6 = 36	36

L'indice I est augmenté de 1 après chaque parcours de la boucle. C'est ainsi que nous pouvons atteindre chaque élément du tableau. En même temps, nous multiplions chaque fois I par lui-même et nous affectons le résultat à la variable d'indice I.
C'est le même principe, l'utilisation du compteur pour calculer les valeurs des différents éléments du tableau, qui a été utilisé pour l'exemple b.

Exemple b)

Dans cet exemple, les différents éléments du tableau diminuent de 2 en partant de 10 chaque fois que l'indice augmente de 1. Nous ne pouvons pas utiliser directement l'indice pour obtenir ces valeurs:

FOR I=10 TO 0 STEP -2 **FAUX!**

car dans ce cas, nous arriverions à un tout autre résultat : chaque élément du tableau recevrait en effet une valeur égale à son indice et nous aurions un tableau comportant les éléments 10 à -2. Il nous faut donc trouver une formule qui affecte à chaque élément du tableau d'indice 1 à 6 les valeurs 10 à 0 :

```
10 DIM X(6)
20 FOR I=1 TO 6
30 X(I)=12 - 2*I
40 NEXT I
50 END
```

Nous avons placé la formule en ligne 30. Vous voyez que nous avons une fois encore utilisé l'indice pour calculer la valeur des différents éléments. Vous pouvez contrôler la justesse de notre formule en complétant le programme avec les lignes suivantes qui sortiront le contenu de tous les éléments du tableau :

```
50 FOR I=1 TO 6
60 PRINT X(I)
70 NEXT I
80 END
```

Si vous avez du mal à comprendre d'emblée le fonctionnement de notre formule, vous pouvez utiliser la méthode que nous avons employée pour l'exemple précédent et écrire sur le papier ce que donnera cette formule pour chaque valeur de l'indice I. Voyons maintenant l'exemple c) où vous devriez également avoir décelé une certaine régularité.

Exemple c)

Si cette suite de nombres vous semble familière, c'est qu'il s'agit en effet des puissances de 2 que nous avons déjà rencontrées dans le chapitre consacré aux systèmes numériques. Il va donc être très facile de trouver une formule de calcul des différentes valeurs des éléments de notre tableau à partir de la valeur du compteur I :

```
10 DIM X(6)
20 FOR I=1 TO 6
30 X(I)=2^I
40 NEXT I
50 END
```

Vous pouvez ici aussi contrôler la justesse de la formule en ajoutant les lignes de programme que nous avons utilisées à l'exemple précédent. Vous pouvez également écrire sur le papier le résultat de la formule pour les valeurs successives de I.

Cette technique de calcul des valeurs d'un tableau à partir de la valeur d'un indice est très importante. Assurez-vous donc que vous l'avez parfaitement comprise. Vous n'aurez plus dès lors de difficulté à comprendre la logique de programmes plus complexes qui utilisent cette technique.

Nous n'avons jusqu'ici pris comme exemples que des tableaux numériques. En ce qui concerne les tableaux alphanumériques, leur contenu ne présente pas en principe ce caractère de régularité qui est parfaitement possible dans les tableaux numériques. Le contenu des différents éléments d'un tableau alphanumérique est souvent entré au moyen du clavier. Un tableau alphanumérique peut également être "chargé" à partir de lignes de DATA et de l'instruction READ.

Les tableaux alphanumériques sont utilisés par exemple pour stocker dans la mémoire de l'ordinateur des noms, des adresses ou des valeurs numériques (sous forme de chaînes de caractères).
L'utilisation des tableaux alphanumériques est donc plus variée que celle des tableaux numériques.

Nous allons créer un petit programme d'exemple qui nous permette d'entrer dans l'ordinateur les prénoms de nos amis. Bien sûr ce programme sera sans intérêt pour le moment puisque nous ne savons pas encore comment sauvegarder ces données sur un périphérique de stockage. Mais cet exemple nous sera très utile pour mieux comprendre plus tard les programmes de gestion de fichier.

Comme vous n'avez certainement pas en tête le nombre précis d'amis dont vous voudriez entrer le nom, ce type de programme ne doit pas faire appel à l'utilisateur pour dimensionner le tableau mais lui indiquer simplement quand il dépasse la capacité de ce tableau. Voici comment un tel programme pourrait être construit :

```
5   CLEARW 2:FULLW 2
10  REM Liste de prenoms
20  DIM Y$(6)
30  Z=Z+1
40  INPUT"Prenom";Y$(Z)
50  PRINT"Autres entrees o/n ?"
60  A$=CHR$(INP(2))
70  IF A$<>"o" THEN 100
80  IF Z<6 THEN 30
90  PRINT"Liste pleine!"
100 END
```

Nous avons pour plus de clarté choisi de n'affecter que 6 éléments à notre tableau (ligne 20). La ligne 30 contient le compteur qui est augmenté de 1 lors de chaque parcours. Comme nous ne connaissons pas le nombre exact de prénoms que nous voulons entrer, nous ne pouvons pas utiliser une boucle FOR ... NEXT. La ligne 40 demande donc avec INPUT l'entrée d'un prénom et l'affecte à l'élément du tableau d'indice Z. La ligne 50 demande si d'autres entrées doivent être faites. La ligne 70 compare le caractère entré à "o"; si ce caractère est différent de "o", le programme saute à la ligne 100 qui met fin au programme. Si ce caractère est "o", la ligne 80 compare le compteur à 6. Si le compteur a déjà atteint la valeur 6, la ligne 90 sort le message "Liste pleine" et le programme se termine. Si nous n'avions pas prévu ce test sur la valeur du compteur Z, le programme serait interrompu après la septième entrée par le message d'erreur :

Subscript refers to element outside the array at line 40

Si en effet Z atteint la valeur 7, la ligne 40 tente d'appeler l'élément Y$(7) qui n'existe pas d'après l'instruction DIM de la ligne 20. Il est plus "propre" d'essayer d'éviter de telles interruptions de programme. Vous auriez toutefois pu obtenir le même résultat avec ON ERROR GOTO et une routine de traitement des erreurs appropriée.

Ce programme n'est bien sûr pas particulièrement performant. Vous pourriez l'améliorer en demandant en fin de programme s'il faut sortir la liste de tous les prénoms entrés. Vous pouvez utiliser une instruction IF ... THEN pour cette demande et une boucle FOR ... NEXT pour sortir les différents éléments du tableau.

Vous pouvez également augmenter la puissance de votre programme, par exemple avec l'instruction DIM Y$(200), ce qui sera certainement suffisant pour une gestion de données personnelle. Mais si vous voulez pouvoir entrer d'autres données que les prénoms de vos amis, un tableau unidimensionnel ne suffira pas.

Voyons maintenant un exemple de création d'un tableau avec les instructions READ et DATA. Supposons par exemple que nous ayons souvent besoin dans un programme d'utiliser les noms des jours de la semaine. Il est inutile d'entrer les noms de la semaine chaque fois que vous lancerez votre programme puisque vous pouvez les placer dans des lignes de DATA et les faire lire au début du programme avec l'instruction READ :

```
5  CLEARW 2:FULLW 2
10 REM Jours de la semaine
20 DIM jours$(7)
30 FOR I=1 TO 7
40 READ jours$(I)
50 NEXT I
60 DATALundi,Mardi,Mercredi,Jeudi,Vendredi,Samedi,Dimanche
70 REM Sortie oui/non
80 PRINT"Sortie du tableau o/n ?"
90 A$=CHR$(INP(2))
```

```
100 IF A$<>"o" THEN 140
110 FOR I=1 TO 7
120 PRINT jours$(I)
130 NEXT I
140 END
```

Ce programme ressemble beaucoup au programme permettant d'entrer une liste de prénoms. La différence essentielle est la ligne 40 qui lit les différents éléments du tableau non pas avec une instruction INPUT mais avec une instruction READ. La fin du programme permet à l'utilisateur de faire sortir la totalité du tableau.

Voici maintenant des exercices qui doivent vous permettre de vous assurer que vous avez bien compris le principe et l'utilisation des tableaux à une dimension, avant que nous n'abordions les tableaux à plusieurs dimensions. Bonne chance !

EXERCICES

1) Ecrivez un programme qui entre 6 noms et les place dans un tableau. Le programme doit également sortir le premier nom dans l'ordre alphabétique. Essayez ce programme avec les noms Raoul, Pierre, Henri, Christine, France et Hyacinthe. N'oubliez pas que vous pouvez comparer les chaînes de caractères entre elles. Le programme devra ici sortir "Christine".

2) Ecrivez un programme qui génère 6 nombres aléatoires compris entre 50 et 150 et les place dans un tableau. Le programme devra également sortir le plus grand de ces nombres.

3) Soit le tableau suivant X(6) :

X(1)	X(2)	X(3)	X(4)	X(5)	X(6)
0	2	6	12	20	30

Ecrivez un programme avec une formule qui permette de créer ce tableau. Faites sortir le tableau pour pouvoir contrôler la justesse de la formule.

4.1.3 LES TABLEAUX A PLUSIEURS DIMENSIONS

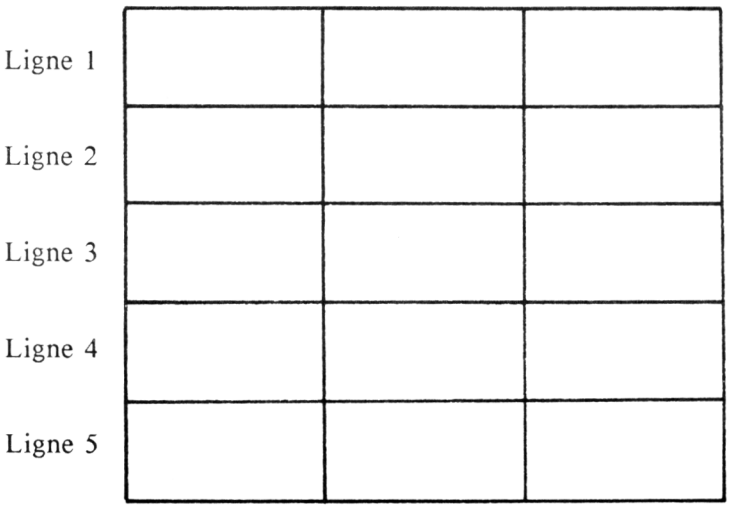
Nous n'avons utilisé jusqu'ici que des tableaux à une dimension. Si nous comparons les tableaux à des listes, il est cependant évident que les listes utilisées dans la vie courante comportent le plus souvent plusieurs informations à chaque ligne. Elles se composent donc de LIGNES et de COLONNES. Supposez que vous vouliez par exemple étendre le programme qui place les prénoms dans un tableau, de façon à ce que les noms et les dates de naissance correspondant à chaque prénom puissent être appelées avec le même indice.

Rien de plus facile, direz-vous : il suffit de créer trois tableaux : A$(X) pour les prénoms, B$(X) pour les noms de famille et C$(X) pour les dates de naissance. Nous pourrions donc de manière générale créer un tableau différent pour chaque rubrique d'une liste de données. Mais nous pourrions alors faire les mêmes remarques qu'avant que nous ne commencions à utiliser les tableaux. Différents tableaux seraient lourds à gérer, surtout si le nombre de rubriques devenait important. Il existe bien sûr une solution plus pratique :

les tableaux à plusieurs dimensions.

Nous avons dans notre exemple besoin d'un tableau à deux dimensions pour pouvoir affecter les données de nom et de date de naissance à la même ligne que le prénom correspondant. Il nous faut donc bien un tableau structuré en lignes et en colonnes, comme le montre l'illustration suivante.

NOM DU TABLEAU = A$



L'instruction DIM permettant de dimensionner ce tableau sera :

DIM A$(5,3)

Cette instruction génère donc un tableau de 5 lignes sur 3 colonnes. Si vous n'utilisez pas l'instruction DIM mais que vous utilisiez simplement un des éléments de ce tableau, l'ordinateur exécutera automatiquement l'instruction DIM A$(10,10). Vous pouvez donc vous passer de l'instruction DIM tant que votre tableau ne comprend pas plus de 11 lignes ni plus de 11 colonnes. Mais si votre tableau est plus petit, la place perdue du fait du dimensionnement automatique à 11 lignes sur 11 colonnes peut être assez considérable. Il vaut donc mieux, même quand elle n'est pas indispensable, utiliser l'instruction DIM.

Voyons maintenant comment utiliser un tableau à deux dimensions. Supposons par exemple que vous vouliez remplir de données les trois colonnes de la première ligne de notre tableau. Vous pourriez utiliser une ligne de programme telle que celle-ci :

```
40 INPUT"Prenom,Nom,ne le";A$(1,1),A$(1,2),A$(1,3)
```

Cette ligne remplit bien sa fonction puisqu'elle permet d'entrer trois données qu'elle affecte à trois éléments de notre tableau. Mais pour plus de clarté, il vaudrait mieux écrire :

```
40 INPUT"Prenom";A$(1,1)
50 INPUT"Nom";A$(1,2)
60 INPUT"Ne le";A$(1,3)
```

Vous pourriez bien sûr vous demander pourquoi nous ne faisons pas entrer ces trois données au moyen d'une boucle, maintenant que nous avons un tableau à deux dimensions qui rend cela parfaitement possible. En l'occurrence, toutefois, nous ne pourrions utiliser une boucle puisque nous avons besoin de sortir un texte de commentaire correspondant à chacune des trois données à entrer. Nous garderons donc nos trois instructions INPUT. Mais, comme nous voulons entrer les données de prénom, nom et date de naissance pour 6 personnes différentes, nous pouvons fort bien utiliser une boucle FOR ... NEXT qui nous permette de répéter 6 fois la procédure des lignes 40 à 60 :

```
10 REM Entrer 6 prenoms,
20 REM noms et dates de naissance
30 DIM A$(6,3)
40 FOR I=1 TO 6
50 INPUT"Prenom";A$(I,1)
60 INPUT"Nom";A$(I,2)
70 INPUT"Ne le";A$(I,3)
80 NEXT I:END
```

Vous trouverez des routines semblables dans nombre de petits programmes de gestion de fichier. Les tableaux à plusieurs dimensions peuvent cependant être remplis non seulement à partir du clavier, mais aussi évidemment à partir de lignes de DATA lues par l'instruction READ. Vous connaissez déjà cette technique que nous avons utilisée pour les tableaux à une dimension. Lorsqu'il s'agit d'un tableau à plusieurs dimensions, nous pouvons utiliser des boucles FOR ... NEXT imbriquées. L'exemple suivant vous montre comment on pourrait par exemple remplir de données figurant en lignes de DATA un tableau à deux dimensions d'une taille de 3 lignes sur 4 colonnes :

```
10 REM Charger un tableau a partir d'une ligne DATA
20 DIM X(3,4)
30 FOR Z=1 TO 3
40 FOR S=1 TO 4
50 READ X(Z,S)
60 NEXT S,Z
70 DATA 11,12,13,14,21,22,23,24,31,32,33,34
80 REM Sortie du tableau
90 PRINT"Sortir le tableau (o/n) ?"
100 A$=CHR$(INP(2))
110 IF A$<>"o" THEN 150
120 FOR Z=1 TO 3
130 PRINT X(Z,1);X(Z,2);X(Z,3);X(Z,4)
140 NEXT Z
150 END
```

La boucle intérieure (S) remplit de données les divers éléments d'une ligne du tableau. Une fois cette boucle terminée, la boucle extérieure (Z) passe à la ligne suivante, jusqu'à la troisième ligne du tableau. Voici comment on pourrait représenter le remplissage progressif de notre tableau par cet exemple de programme :

TABLEAU A(3,4)

```
11  *  *  *   11 12  *  *   11 12 13  *   11 12 13 14
 *  *  *  *    *  *  *  *    *  *  *  *    *  *  *  *
 *  *  *  *    *  *  *  *    *  *  *  *    *  *  *  *

11 12 13 14   11 12 13 14   11 12 13 14   11 12 13 14
21  *  *  *   21 22  *  *   21 22 23  *   21 22 23 24
 *  *  *  *    *  *  *  *    *  *  *  *    *  *  *  *

11 12 13 14   11 12 13 14   11 12 13 14   11 12 13 14
21 22 23 24   21 22 23 24   21 22 23 24   21 22 23 24
31  *  *  *   31 32  *  *   31 32 33  *   31 32 33 34
```

Si vous voulez faire sortir sur l'écran le tableau que notre programme a créé, vous n'avez qu'à appuyer sur la touche "o". Le tableau sera alors sorti sous la forme que montre l'illustration précédente. La ligne 130 sort en effet en une seule fois les 4 éléments de chaque ligne. Nous n'avons utilisé une boucle que pour sortir les éléments des différentes lignes. Voici toutefois comment nous aurions également pu arriver au même résultat :

```
.

80  REM Sortie du tableau
90  PRINT"Sortir le tableau (o/n) ?"
100 A$=CHR$(INP(2))
110 IF A$<>"o" THEN 150
120 FOR Z=1 TO 3
130 FOR S=1 TO 4
140 PRINT A(Z,S);:ZZ=ZZ+1
150 IF ZZ=4 THEN ZZ=0:PRINT:PRINT
160 NEXT S,Z
170 END
```

Cette nouvelle version du programme utilise deux boucles FOR ... NEXT imbriquées. Le point-virgule en ligne 140 fait que les différents éléments d'une même ligne sont affichés l'un à la suite de l'autre. Mais pour que la structure du tableau en lignes et colonnes apparaisse à l'écran, il faut que l'ordinateur passe à la ligne après avoir sorti les 4 éléments d'une même ligne. C'est pourquoi nous avons utilisé un compteur supplémentaire (ZZ) qui compte combien de fois une sortie avec PRINT a été effectuée. La ligne 150 teste si ce compteur ZZ a déjà atteint la valeur 4. Si c'est le cas, le compteur est réinitialisé à 0 puis une instruction PRINT supplémentaire est exécutée dont l'effet est simplement de faire passer la sortie sur écran à la ligne suivante. Nous avons ajouté une seconde instruction PRINT simplement pour laisser une ligne entre deux lignes de données et rendre la lecture du tableau plus claire.

Reprenons maintenant notre exemple de programme pour entrer des noms,prénoms et dates de naissance. Nous avons dit qu'on ne sait pas en général, lorsqu'on réalise un tel programme, pour combien de personnes on aura de données différentes à entrer. On doit par contre toujours essayer de déterminer de façon précise combien de données et lesquelles on voudra pouvoir enregistrer pour chaque personne. Par exemple, on pourra vouloir enregistrer pour chaque personne le nom, le prénom et le numéro de téléphone. Si vous réalisez donc un programme d'agenda téléphonique, vous ne connaissez en général qu'une des deux dimensions nécessaires de votre tableau, celle qui correspond aux nombres de données à entrer pour chaque personne. L'autre dimension, qui correspond donc au nombre de personnes dont vous voudrez pouvoir enregistrer les coordonnées, doit être estimée grossièrement. Par exemple, si vous pensez que votre cercle de relations ne dépasse pas 100 personnes, une instruction DIM X$(120,3) devrait être largement suffisante. Voici un exemple :

```
5   FULLW 2
10  REM Entree de donnees
20  DIM X$(120,3)
30  CLEARW 2
40  Z=Z+1
50  INPUT"Prenom";X$(Z,1)
60  PRINT
70  INPUT"Nom";X$(Z,2)
80  PRINT
90  INPUT"Ne le";X$(Z,3)
100 PRINT
110 PRINT"Voulez-vous entrer "
120 PRINT"d'autres donnees (o/n) ?"
130 A$=CHR$(INP(2))
140 IF A$<>"o" THEN 30
150 END
```

Ce problème pourrait être résolu de façon plus élégante, mais l'essentiel est pour nous de bien comprendre le principe. Nous sommes obligé de créer un compteur en ligne 40 pour faire varier l'indice des divers éléments du tableau puisque nous n'utilisons pas de boucle FOR ... NEXT. Les données sont ensuite entrées avec des instructions INPUT et affectées aux éléments correspondants du tableau. Si vous voulez entrer d'autres données, le programme saute en ligne 30, sinon il est terminé. Dans un programme complet de gestion de fichier nous aurions ici au lieu de l'instruction END un retour au menu principal pour permettre à l'utilisateur de sélectionner d'autres points du menu. Vous voyez que nous avons utilisé en ligne 140 une comparaison avec IF ... THEN pour demander à l'utilisateur s'il souhaite entrer d'autres données. Retenez donc bien qu'on ne peut utiliser une boucle FOR ... NEXT que lorsqu'on veut fixer d'avance le nombre de données à entrer. Dans le cas contraire, il faut utiliser IF ... THEN.

Ces exemples devraient suffir pour le moment à vous donner une idée claire du maniement des tableaux à plusieurs dimensions.

Notez que l'Atari ST peut admettre théoriquement des tableaux ayant jusqu'à 15 dimensions. Remarquez toutefois que si nous pouvons nous représenter des tableaux de trois dimensions, sous la forme d'un cube par exemple, nous sommes incapable de nous représenter des tableaux de 4 ou 5 dimensions. Cela ne signifie d'ailleurs pas que de tels tableaux, même si nous ne pouvons bien les imaginer, ne puissent pas être utilisés pour résoudre certains problèmes.

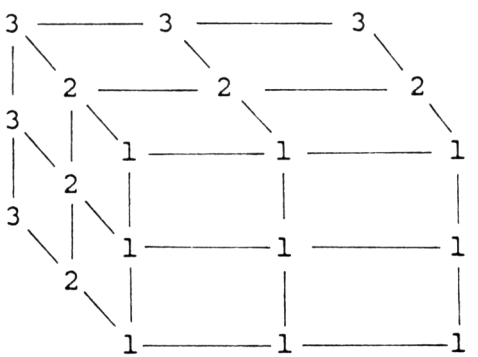
Sans aller jusque là, voici tout de même l'exemple d'un tableau en trois dimensions. Les indices auront la signification suivante :

X = ligne

Y = colonne

Z = profondeur

Nous allons essayer de créer un tableau en trois dimensions qui représente un dé ayant une arête de trois unités. Vous pouvez vous représenter un tel tableau comme trois tableaux à deux dimensions placés l'un derrière l'autre. Avec un peu d'imagination vous pouvez essayer de reconnaître notre dé en perspective sur l'image suivante :



Pour réaliser ce tableau, l'instruction DIM devra avoir le format suivant :

DIM W(X,Y,Z)

Pour notre exemple, nous aurons :

DIM W(3,3,3)

Notre tableau possède ainsi 27 éléments (3*3*3=27) ou plutôt 64 (4*4*4=64) si nous tenons compte de l'élément 0.

Nous vous proposons maintenant l'exercice suivant : écrivez un programme qui remplisse ce tableau de données. Remplissez d'abord tous les éléments d'une ligne, puis toutes les lignes d'une "page" puis toutes les pages du tableau. Essayez de remplir le tableau avec les données qui figurent sur notre illustration. Il est inutile d'essayer d'obtenir un effet de relief. Essayez bien sûr de résoudre ce problème sans regarder la solution.

SOLUTION

```
10 REM Tableau en 3 dimensions
20 DIM W(3,3,3)
30 FOR Z=1 TO 3
40 FOR Y=1 TO 3
50 FOR X=1 TO 3
60 READ W(X,Y,Z)
70 NEXT X,Y,Z
80 DATA 1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,3
90 REM Sortie du tableau
100 FOR Z=1 TO 3
110 FOR Y=1 TO 3
120 FOR X=1 TO 3
130 PRINT W(X,Y,Z);:ZZ=ZZ+1
```

```
140 IF ZZ=3 THEN ZZ=0:PRINT
150 NEXT X,Y,Z
160 END
```

L'instruction de dimensionnement en ligne 20 devient de plus en plus indispensable, chaque fois que nous passons à un tableau de dimension supérieure. En effet, alors que nous n'avons ici besoin que de 27 (ou 64) éléments, l'ordinateur créera automatiquement 11*11*11=1331 éléments s'il rencontre une variable à trois indices qui n'a pas été encore dimensionnée. Il exécutera en effet l'instruction DIM W(10,10,10). L'instruction DIM permet donc d'économiser énormément de place en mémoire. Les lignes 30 à 50 vous montrent une imbrication de trois boucles FOR ... NEXT. La boucle extérieure remplit les différentes pages, la boucle du milieu les différentes lignes et la boucle intérieure les différents éléments de chaque ligne. La ligne 70 ferme les trois boucles avec une seule instruction NEXT. Faites toujours bien attention à fermer les boucles dans le bon sens, c'est-à-dire dans le sens exactement contraire au sens d'ouverture.

Ce chapitre sur les tableaux à plusieurs dimensions touche maintenant à sa fin. Il s'agit là d'une technique très importante et très utilisée en BASIC, notamment dans les programmes de gestion de fichier. Bien entendu seuls les tableaux à une ou deux dimensions sont véritablement d'usage courant.

Si vous avez bien assimilé l'utilisation des tableaux, vous pouvez maintenant aborder le chapitre suivant consacré aux sous-programmes qui permettent de gagner du temps de programmation ainsi que de la place en mémoire lorsqu'une même suite d'instructions doit être utilisée plusieurs fois dans le même programme.

4.2 LES SOUS-PROGRAMMES

Un sous-programme est une suite d'instructions ou de lignes d'instructions qui est utilisée plusieurs fois dans un même programme et qu'on va donc pouvoir appeler de différents endroits du programme, chaque fois que cette suite d'instructions devra être exécutée. On place généralement les différents sous-programmes les uns à la suite des autres en début ou en fin de programme. Vous pouvez appeler un sous-programme avec l'instruction :

GOSUB (numéro de ligne)

GOSUB signifie "GO to SUBroutine" (va à la sous-routine).
Lorsque le programme rencontre cette instruction, il note le numéro de la ligne où elle figure. Il saute alors au sous-programme et continue l'exécution du programme à partir du début du sous-programme, jusqu'à ce qu'il rencontre l'instruction :

RETURN

L'ordinateur continue alors l'exécution du programme à partir de la première instruction qui figure à la suite de l'instruction GOSUB. Si le programme rencontre une instruction RETURN sans avoir rencontré l'instruction GOSUB auparavant, il s'interrompt en sortant le message d'erreur :

RETURN statement needs matching GOSUB at line (n° de ligne)

Cela signifie que vous ne pouvez appeler un sous-programme qu'avec l'instruction GOSUB. Si vous faites l'erreur de sauter à un sous-programme avec l'instruction GOTO, l'ordinateur ne notera pas où il a rencontré l'instruction GOTO et lorsqu'il rencontrera l'instruction RETURN qui met fin au sous-programme, il ne saura plus où aller et sortira le message d'erreur **Unexpected RETURN**. Voici un exemple de ce qu'il ne faut pas faire :

```
110 IF A<1 THEN GOTO 130   !!! E R R E U R !!!
120 GOTO 90
130 REM SOUS-PROGRAMME
140 A=A+1
150 RETURN
```

Si A est plus petit que 1, le programme sautera ici en 130. Il exécutera la ligne 140 puis lorsqu'il rencontrera RETURN, il sortira le message d'erreur :

RETURN statement needs matching GOSUB at line 150

et interrompra le programme. Il aurait fallu écrire en ligne 110 :

```
110 IF A<1 THEN GOSUB 130
```

Le programme suivant illustre une autre erreur favorite des programmeurs de sous-programmes, qui de plus est assez difficile à dépister :

```
10  REM ERREUR AVEC UN SOUS-PROGRAMME
20  PRINT
30  PRINT Z
40  GOSUB 70
50  Z=Z+1:GOTO 20
60  END
70  REM SOUS-PROGRAMME
80  FOR I=1 TO 25
90  PRINT I;
100 IF I >= 15 THEN 50
110 NEXT I
120 RETURN
```

Si vous lancez ce programme, vous constaterez qu'après avoir parcouru 15 ou 16 fois le sous-programme, le déroulement du programme sera interrompu avec le message d'erreur :

?You have nested subroutine calls too deep at line 40

L'erreur ne vient pas ici de l'appel du sous-programme. Le programme commence en effet par vider l'écran puis par constituer une ligne d'espaces (ligne 20). La valeur actuelle du compteur Z qui compte le nombre d'appels du sous-programme est ensuite sortie. La ligne 40 appelle alors le sous-programme avec GOSUB 70. Le programme saute au sous-programme en ligne 70 et commence par exécuter la boucle FOR ... NEXT. La ligne 90 sort la valeur de la variable compteur I.

La ligne 100 contient ensuite 2 erreurs en une seule: premièrement il ne faut jamais quitter une boucle FOR ... NEXT avec GOTO sous peine de provoquer des problèmes dans la gestion de ces boucles par l'ordinateur. Deuxièmement et surtout, il ne faut pas sortir d'un sous-programme autrement qu'en utilisant l'instruction RETURN. Or la ligne 100 saute du sous-programme à la ligne 50 si I est supérieur ou égal à 15. Il aurait donc fallu écrire THEN RETURN au lieu de THEN 50.

Vous pouvez bien sûr utiliser autant d'instructions GOTO que vous voulez à l'intérieur d'un sous-programme mais vous ne devez pas quitter un sous-programme ni entrer dans un sous-programme avec l'instruction GOTO. Ce type d'erreur est d'ailleurs facile à éviter avec un organigramme clair.

Essayons maintenant d'appliquer la technique des sous-programmes à un exemple concret. Reprenons pour cela le programme de cours de calcul. Si vous étudiez le listing de ce programme, vous constaterez vite que certaines parties du programme se répètent plusieurs fois. On pourrait donc tout à fait remplacer ces parties du programme par un sous-programme. Voici notamment les lignes qui se répètent souvent :

```
230 CLEARW 2
240 PRINT TAB(10)"Entrez le plus grand nombre"
250 PRINT
260 PRINT TAB(10)"pour l'addition."
270 PRINT
290 PRINT TAB(10);:INPUT"Plus grand nombre";GR
299 REM
300 REM GENERATION DE NOMBRES ALEATOIRES
301 REM
310 A1=INT(GR*RND(1))+1
320 A2=INT(GR*RND(1))+1
329 REM
330 REM CALCUL DU RESULTAT
331 REM
340 ER=A1+A2
350 CLEARW 2
360 PRINT
370 PRINT A1 "+" A2 "=";
380 INPUT ES
390 IF ES=ER THEN PRINT:PRINT TAB(10)"C'EST JUSTE!":
    F=0:GOTO 450
400 PRINT:PRINT TAB(10)"C'EST FAUX!"
410 FOR I=0 TO 2000:NEXT I
420 F=F+1
430 IF F<=2 THEN 350
440 PRINT
450 FOR I=0 TO 2000:NEXT I
460 PRINT TAB(5) "Le résultat est";ER
470 FOR I=0 TO 3000:NEXT I
480 PRINT TAB(5) "Encore un exercice O/N";
490 INPUT A$
500 IF A$="O" THEN F=0:GOTO 300
510 GOTO 10
```

Ces lignes de programme qui figurent à 4 reprises dans notre programme pourraient être regroupées en différents blocs. Un premier bloc pourrait être constitué par les lignes 230 à 290 car l'entrée d'un nombre maximum est demandée pour chaque type de calcul. Le seul problème est que le message demandant à l'utilisateur d'entrer le nombre maximum doit pouvoir être adapté en fonction de chaque type de calcul. Le message :

Entrez le plus grand nombre

pour l'addition

doit donc pouvoir être modifié par le sous-programme en fonction du type de calcul.

Souvenons-nous par ailleurs que le programme saute au type de calcul choisi par l'utilisateur avec l'instruction ON X GOTO. Nous pourrions donc utiliser cet X comme indice du type de calcul. Nous pouvons ainsi créer en début de programme un tableau qui contienne les termes d'addition, soustraction, division et multiplication, dans l'ordre dans lequel ces différentes opérations figurent dans le menu. Nous pourrions même alors faire éditer le menu grâce à ce tableau. Voici comment se présenterait le début du programme ainsi modifié :

```
10 REM    ************************
20 REM    * DEBUT DU PROGRAMME *
30 REM    ************************
40 REM
50 DIM calcul$(4),operateur$(4)
60 FOR I=1 TO 4
70 READ calcul$(I),operateur$(I)
80 NEXT I
90 DATA l'addition,+,la soustraction,-
,division,/,multiplication,*
100 GOTO 580
.
```

La ligne 50 génère deux tableaux calcul$ et operateur$ (n'oubliez pas que vous pouvez utiliser avec l'Atari ST des noms de variables ou tableaux très explicites). La boucle FOR ... NEXT en ligne 60 charge les données en DATA dans les deux tableaux en affectant à calcul$ les noms de type de calcul et à operateur$ les signes correspondant à chaque type d'opération. Nous verrons plus tard pourquoi le programme saute de la ligne 100 à la ligne 580. Voyons maintenant comment nous pouvons utiliser les tableaux que nous venons de créer pour sortir le menu sur l'écran. Voici donc comment se présente maintenant le menu :

```
.

570 REM ***************
580 REM *     MENU     *
590 REM ***************
600 CLEARW 2:F=0
610 PRINT
620 PRINT TAB(12)"COURS DE CALCUL"
630 PRINT:PRINT
640 PRINT TAB(12)"Choisissez:"
650 PRINT
660 PRINT TAB(12)"1 pour "calcul$(1)
670 PRINT
680 PRINT TAB(12)"2 pour "calcul$(2)
690 PRINT
700 PRINT TAB(12)"3 pour "calcul$(3)
710 PRINT
720 PRINT TAB(12)"4 pour "calcul$(4)
730 PRINT
740 PRINT TAB(12)"5 pour FIN"
750 PRINT
760 PRINT TAB(12)"Quel nombre"
770 E$=CHR$(INP(2))
780 IF VAL(E$)<1 OR VAL(E$)>5 THEN 770
790 P=VAL(E$):ON P GOTO 800,890,990,1090,1180
.
```

Nous aurions aussi bien pu créer le menu à l'aide d'une boucle, comme dans l'exemple ci-dessous :

```
.
.

660 FOR I=1 TO 4
670 PRINT TAB(12);I;"pour ";calcul$(I)
680 PRINT
690 NEXT I
.
.
```

Mais nous y avons renoncé pour conserver une présentation plus claire au programme.
Dans ce nouveau menu, la ligne 770 lit un caractère et la ligne 780 teste si ce caractère est "recevable". Si c'est bien le cas, la valeur numérique de ce caractère est calculée avec VAL(E$) et affectée à la variable P. Le programme saute ensuite à différentes parties du programme, en fonction de la valeur de P. Pour l'addition par exemple, vous entrerez 1, P vaudra 1 et le programme sautera à la ligne 800. C'est en effet en 800 que commence la partie du programme, le "module" de l'addition. Voyons comment il se présente :

```
.

800 REM **********
810 REM  ADDITION
820 REM **********
830 GOSUB 110
840 GOSUB 310
850 ER=A1+A2
860 GOSUB 390
870 IF A$="o" THEN 840
880 GOTO 580
.
```

La ligne 830 appelle maintenant le premier sous-programme. Ce sous-programme permet l'entrée du nombre le plus grand devant être utilisé dans le calcul. Voici comment il se présente maintenant :

```
.
.

110 REM ******************
120 REM * SOUS-PROGRAMMES *
130 REM ******************
140 REM ****************************
150 REM * Entree du plus grand nombre *
160 REM ****************************
170 CLEARW 2:A$="":B$=""
180 PRINT TAB(10)"Entrez le plus grand nombre"
190 PRINT
200 PRINT TAB(10)"pour "calcul$(P)
210 PRINT
220 PRINT TAB(10)"Plus grand nombre ?";
230 FOR I=1 TO 3
240 A$=CHR$(INP(2))
250 IF ASC(A$)<48 OR ASC(A$)>57 THEN 240
260 B$=B$+A$:PRINT A$;
270 NEXT I
280 GR=val(B$)
290 RETURN
.
.
```

L'instruction CLEARW 2 en ligne 170 vous est certainement déjà devenue familière. Mais la deuxième partie de cette ligne mérite un commentaire car elle est typique pour un sous-programme. Les variables A$ et B$ sont en effet annulées, on leur affecte une chaîne vide pour être sûr que le dispositif de la ligne 260 fonctionnera bien, même si ce sous-programme est appelé à plusieurs reprises, c'est-à-dire pour plusieurs types d'opérations différentes. En effet B$ reçoit en 260 le nombre que vous entrez.

Une fois que le programme sortira du sous-programme, B$ gardera ce nombre intact. Si donc le programme saute à nouveau au même sous-programme pour un autre type d'opération, le nombre que vous entrerez cette fois ira s'ajouter aux chiffres que B$ aura reçu la première fois. Il faut donc toujours penser à réinitialiser les variables de ce type au début d'un sous-programme.

Selon une technique qui nous est maintenant familière, nous entrons les chiffres composant le plus grand nombre en ligne 240 et la ligne 250 contrôle qu'il ne s'agit que de chiffres entre 0 et 9. PRINT A$; en ligne 260 sort le caractère que l'utilisateur vient d'entrer et lui permet ainsi de voir ce qu'il entre. Notre programme n'admet que des nombres de trois chiffres. Si vous voulez entrer un nombre de deux chiffres, vous devez entrer d'abord un 0 puis vos deux chiffres.

La ligne 200 vous montre pourquoi nous avons créé un tableau contenant les noms des opérations: cela nous permet en effet d'adapter le message demandant l'entrée du plus grand nombre en fonction du type d'opération fourni par P.

La ligne 840 du programme d'addition appelle alors le sous-programme de génération de nombres aléatoires qui est le sous-programme le plus petit et le plus simple :

```
.

300 REM
310 REM GENERATION DE NOMBRES ALEATOIRES
320 REM
330 A1=INT(GR*RND(1))+1
340 A2=INT(GR*RND(1))+1
350 RETURN
.
.
```

Voici maintenant le sous-programme de création de l'exercice :

```
360 REM
370 REM CREATION D'UN EXERCICE
380 REM
390 CLEARW 2
400 PRINT
410 PRINT A1;operateur$(P);A2 "=";
420 INPUT ES
430IFES=ERTHENPRINT:PRINTTAB(10)"C'ESTJUSTE!":F=0:GOTO500
440 PRINT:PRINT TAB(10)"C'EST FAUX!"
450 FOR I=0 TO 2000:NEXT I
460 F=F+1
470 IF F<=2 THEN 390
480 PRINT
490 FOR I=0 TO 2000:NEXT I
500 PRINT TAB(5) "Le résultat est";ER
510 F=0
520 FOR I=0 TO 3000:NEXT I
530 PRINT
540 PRINT TAB(5) "Encore un exercice O/N";
550 INPUT A$
560 RETURN
.
.
```

Nous nous arrêterons ici uniquement sur les lignes méritant un commentaire particulier. D'abord la ligne 410 : comme nous l'avons fait dans le sous-programme d'entrée du plus grand nombre, nous utilisons ici un tableau pour créer un message pour l'utilisateur. Ce message pourrait être en effet symbolisé ainsi :

A1 + (ou - ou / ou *) A2 = ?

L'opérateur qui convient est sélectionné en fonction du type d'opération P. Nous avons donc généralisé la formulation du message à l'utilisateur pour lui présenter un exercice. Ce faisant nous avons été obligé de fixer une fois pour toutes les variables opérandes : le premier opérande sera toujours A1, le second toujours A2.

Ceci va nous conduire à modifier quelque peu la partie du programme consacrée à la division puisque la formule d'exercice choisie pour la division n'était pas A1 / A2 mais ER / A1.

Vous voyez que les sous-programmes peuvent entraîner et entraînent effectivement souvent des adaptations du programme. Le programme doit être pensé dès le départ de façon à permettre d'isoler un maximum de tâches générales qui pourront être regroupées dans des sous-programmes. Il faut ensuite se demander pour chaque partie du programme appelant un sous-programme quelle préparation doivent subir telles ou telles variables pour que le sous-programme donne bien les résultats attendus.

Nous en avons fini avec l'étude de la partie sous-programmes de notre programme. Reste un petit problème que nous voudrions évoquer. Vous avez certainement remarqué que nous avons placé les sous-programmes au début de notre nouvelle version du programme alors que nombre d'ouvrages recommandent de placer les sous-programmes en fin de programme. En effet, si vous voulez constituer une bibliothèque de sous-programmes, il peut être très pratique d'en placer en 50000, en 55000 etc... et de les ajouter ensuite à la suite de vos programmes avec l'instruction MERGE.

Voyons cependant les arguments qui parlent en faveur de sous-programmes placés en début de programme : on peut objecter que peu importe finalement que ajoutiez un sous-programme à votre programme principal ou que vous ajoutiez votre programme principal à la suite d'un sous-programme. D'autant que l'instruction RENUM vous permet de jongler avec les numéros de ligne de votre sous-programme ou de votre programme. Jusqu'ici donc, match nul.

Mais si on étudie le fonctionnement de votre ordinateur, on s'aperçoit que lorsque vous appelez un sous-programme, l'ordinateur cherche le numéro de la ligne à laquelle il doit sauter en examinant le programme du début. Si donc vous placez les sous-programmes au début du programme, vous diminuez son temps de recherche et donc aussi le temps d'exécution du programme.

Voyons maintenant quelles adaptations le reste du programme va devoir subir pour s'adapter à l'utilisation des sous-programmes. Il s'agit en effet de préparer les variables A1 et A2 pour l'exercice dont nous avons vu qu'il se présentera toujours sous la forme A1 (opérateur) A2 = ?. En fait seules la soustraction et la division nécessitent des adaptations particulières.

En ce qui concerne la soustraction, il faut que A1 soit toujours plus grand que A2. C'est ce que garantissait en fait déjà la ligne 940 qui échange entre elles les valeurs de A1 et A2 lorsque A1 est inférieur à A2. En fait, cette adaptation existait déjà dans la première version de notre programme.

Pour la division par contre nous allons être obligé de modifier quelque peu la logique antérieure du programme. L'exercice de division se présentait en effet ainsi :

ER / A1 = ?

Or nous avons réalisé un sous-programme général d'exercice auquel la division doit également se plier. L'exercice de division doit donc obligatoirement pouvoir se présenter ainsi :

A1 / A2 = ?

Et ER doit être le résultat de cet exercice. C'est ce qui sera réalisé avec les lignes supplémentaires suivantes :

```
.

1040 ER=A1*A2
1050 I=ER:ER=A1:A1=I
.
.
```

Nous calculons toujours le résultat grâce à une multiplication mais nous réalisons en ligne 1050 un échange entre les deux variables A1 et A2 de façon à ce qu'elles soient correctement préparées pour le sous-programme "création d'un exercice". Nous utilisons pour ce faire à nouveau la technique du stockage provisoire : I reçoit provisoirement la valeur de la variable ER. ER reçoit la valeur de A1 qui reçoit celle de I, donc la valeur antérieure de ER. Nos trois variables A1, A2 et ER sont maintenant prêtes et le sous-programme de création d'un exercice donnera bien les résultats escomptés.

Voici maintenant le listing complet de la nouvelle version de notre programme :

```
10    REM    **********************
20    REM    * DEBUT DU PROGRAMME *
30    REM    **********************
40    REM
50    DIM calcul$(4),operateur$(4)
60    FOR I=1 TO 4
70    READ calcul$(I),operateur$(I)
80    NEXT I
90    DATA l'addition,+,la soustraction,-
      ,division,/,multiplication,*
100   GOTO 580
110   REM ******************
120   REM * SOUS-PROGRAMMES *
130   REM ******************
140   REM ******************************
150   REM * Entree du plus grand nombre *
160   REM ******************************
170   CLEARW 2:A$="":B$=""
180   PRINT TAB(10)"Entrez le plus grand nombre"
190   PRINT
200   PRINT TAB(10)"pour "calcul$(P)
210   PRINT
```

```
220  PRINT TAB(10)"Plus grand nombre ?";
230  FOR I=1 TO 3
240  A$=CHR$(INP(2))
250  IF ASC(A$)<48 OR ASC(A$)>57 THEN 240
260  B$=B$+A$:PRINT A$;
270  NEXT I
280  GR=val(B$)
290  RETURN
300  REM
310  REM GENERATION DE NOMBRES ALEATOIRES
320  REM
330  A1=INT(GR*RND(1))+1
340  A2=INT(GR*RND(1))+1
350  RETURN
360  REM
370  REM CREATION D'UN EXERCICE
380  REM
390  CLEARW 2
400  PRINT
410  PRINT A1;operateur$(P);A2 "=";
420  INPUT ES
430  IF ES=ER THEN PRINT:PRINT TAB(10)"C'EST JUSTE!"
     :F=0:GOTO 500
440  PRINT:PRINT TAB(10)"C'EST FAUX!"
450  FOR I=0 TO 2000:NEXT I
460  F=F+1
470  IF F<=2 THEN 390
480  PRINT
490  FOR I=0 TO 2000:NEXT I
500  PRINT TAB(5) "Le résultat est";ER
510  F=0
520  FOR I=0 TO 3000:NEXT I
530  PRINT
540  PRINT TAB(5) "Encore un exercice O/N";
550  INPUT A$
560  RETURN
570  REM **********
```

```
580  REM *      MENU      *
590  REM ***************
600  SCNCLR:F=0
610  PRINT
620  PRINT TAB(12)"COURS DE CALCUL"
630  PRINT:PRINT
640  PRINT TAB(12)"Choisissez:"
650  PRINT
660  PRINT TAB(12)"1 pour "calcul$(1)
670  PRINT
680  PRINT TAB(12)"2 pour "calcul$(2)
690  PRINT
700  PRINT TAB(12)"3 pour "calcul$(3)
710  PRINT
720  PRINT TAB(12)"4 pour "calcul$(4)
730  PRINT
740  PRINT TAB(12)"5 pour FIN"
750  PRINT
760  PRINT TAB(12)"Quel nombre"
770  E$=CHR$(INP(2))
780  IF VAL(E$)<1 OR VAL(E$)>5 THEN 770
790  P=VAL(E$):ON P GOTO 800,890,990,1090,1180
800  REM **********
810  REM  ADDITION
820  REM **********
830  GOSUB 110
840  GOSUB 310
850  ER=A1+A2
860  GOSUB 390
870  IF A$="o" THEN 840
880  GOTO 580
890  REM **************
900  REM  SOUSTRACTION
910  REM **************
920  GOSUB 110
930  GOSUB 310
940  IF A1 < A2 THEN I=A1:A1=A2:A2=I
```

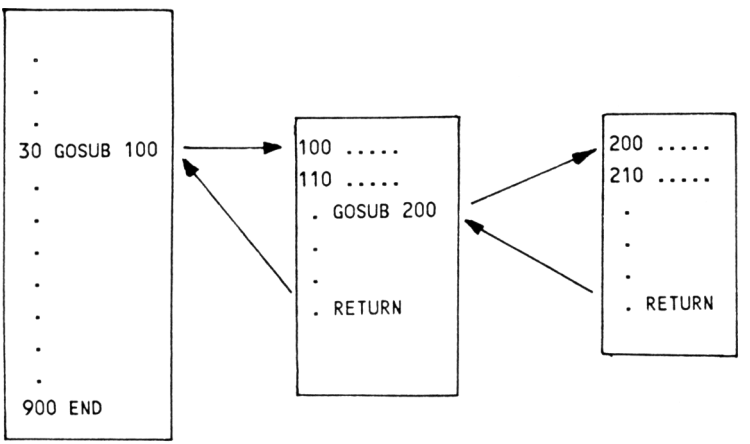
```
950  ER=A1-A2
960  GOSUB 390
970  IF A$="o" THEN 930
980  GOTO 580
990  REM **********
1000 REM  DIVISION
1010 REM **********
1020 GOSUB 110
1030 GOSUB 310
1040 ER=A1*A2
1050 I=ER:ER=A1:A1=I
1060 GOSUB 390
1070 IF A$="o" THEN 1030
1080 GOTO 580
1090 REM ***************
1100 REM  MULTIPLICATION
1110 REM ***************
1120 GOSUB 110
1130 GOSUB 310
1140 ER=A1*A2
1150 GOSUB 390
1160 IF A$="o" THEN 840
1170 GOTO 580
1180 REM *******
1190 REM * FIN *
1200 REM *******
1210 CLEARW 2
1220 END
```

Nous avons donc ainsi économisé environ 43 lignes de programme grâce à l'utilisation de trois sous-programmes et bien que nous ayons ajouté des lignes de commentaires REM! La technique des sous-programmes est très pratique parce qu'elle permet de gagner du temps lors de la programmation et de la place en mémoire. Vous avez certainement compris maintenant pourquoi la ligne 100 contient l'instruction GOTO 580: le programme doit en effet sauter direct-

ement au menu, en sautant par dessus les sous-programmes.
Nous avons déjà expliqué pour quelles raisons nous pensons qu'il est préférable de placer les sous-programmes en début de programme.

Les sous-programmes, comme les boucles, peuvent être imbriqués : un sous-programme peut appeler un autre sous-programme, ce que l'on pourrait symboliser ainsi :

PROGRAMME PRINCIPAL SOUS-PROGRAMME 1 SOUS-PROGRAMME 2



Le programme rencontrera donc l'instruction GOSUB en ligne 30. Il sautera alors au sous-programme 1 en ligne 100. Le sous-programme 1 appellera ensuite le sous-programme 2 avec GOSUB 200.
Le sous-programme 2 sera exécuté jusqu'à l'instruction RETURN. Le programme sortira alors du sous-programme 2 pour retourner au sous-programme 1 qu'il exécutera à partir de l'instruction qui suit directement l'instruction GOSUB. Le sous-programme 1 sera également

exécuté jusqu'à la prochaine instruction RETURN, après quoi le programme retournera au programme principal qu'il continuera d'exécuter à partir de l'instruction qui suit l'instruction GOSUB.

Les sous-programmes peuvent également être appelés avec une instruction qui équivaut pour les sous-programmes à l'instruction ON ... GOTO, l'instruction ON ... GOSUB :

ON P GOSUB 800,890,990

Suivant la valeur de P, le programme sautera au sous-programme 800, 890 ou 990. Si P vaut 0 ou plus de 3, le programme exécutera l'instruction placée à la suite de l'instruction ON ... GOSUB.
Notez simplement qu'une fois le sous-programme exécuté, le programme reprendra l'exécution du programme principal à partir de l'instruction qui suit l'instruction ON ... GOSUB.

Nous espérons que l'explication détaillée de notre exemple de cours de calcul vous aura permis de bien comprendre la technique des sous-programmes. Pour vous permettre de vous en assurer, voici un petit exercice.

Nous n'avons pas en effet épuisé, dans notre dernière version du cours de calcul, les possibilités de la technique des sous-programmes. Essayez donc de regrouper le plus grand nombre possible de lignes dans des sous-programmes. Bien sûr vous devrez adapter quelque peu le programme principal. Nous ne vous demandons pas de placer ces nouveaux sous-programmes en début de programme car cela vous obligerait à trop de travail d'écriture. Essayez de trouver la solution par vous-même. Bonne chance !

SOLUTION

```
.
.
.
780  IF VAL(E$)<1 OR VAL(E$)>5 THEN 770
790  P=VAL(E$)
800  IF P=5 THEN 1100
810  GOSUB 110
820  GOSUB 310
830  ON P GOSUB 880,930,990,1050
840  GOSUB 390
850  IF A$="o" THEN 820
860  GOTO 580
870  REM **********
880  REM  ADDITION
890  REM **********
900  ER=A1+A2
910  RETURN
920  REM **************
930  REM  SOUSTRACTION
940  REM **************
950  IF A1 < A2 THEN I=A1:A1=A2:A2=I
960  ER=A1-A2
970  RETURN
980  REM  **********
990  REM   DIVISION
1000 REM **********
1010 ER=A1*A2
1020 I=ER:ER=A1:A1=I
1030 RETURN
1040 REM ***************
1050 REM  MULTIPLICATION
1060 REM ***************
1070 ER=A1*A2
1080 RETURN
1090 REM *******
```

```
1100 REM * FIN *
1110 REM *******
1120 CLEARW 2
1130 END
```

Nous espérons que vous avez trouvé la solution sans peine. Il y avait en effet 5 lignes de programme qui se répétaient dans les parties du programme consacrées aux différentes opérations :

```
830  GOSUB 110
840  GOSUB 310
860  GOSUB 390
870  IF A$="o" THEN 840
880  GOTO 580
```

Les parties du programme pour chaque type d'opération ne se distinguent donc entre elles que par le calcul proprement dit. C'est pourquoi les lignes 830, 840, 860, 870 et 880 ont été placées à la suite du menu. Un test spécial sur P=5 a été instauré car la fin du programme n'est pas un sous-programme. Les calculs pour les types d'opération ont été alors écrits comme des sous-programmes qui seront appelés avec l'instruction :

ON P GOSUB

Nous avons ainsi économisé encore 9 lignes de programme.

Résumons maintenant les points les plus importants à retenir en ce qui concerne la technique des sous-programmes :

1. Les sous-programmes permettent de regrouper des parties de programme qui se répètent fréquemment.

2. Les sous-programmes sont appelés avec GOSUB et terminés avec RETURN. Ils ne doivent jamais être appelés ou quittés avec l'instruction GOTO. L'instruction GOTO doit être utilisée soit totalement en dehors d'un sous-programme, soit totalement à l'intérieur d'un sous-programme.

3. Les sous-programmes peuvent être imbriqués: un sous-programme 1 peut appeler un sous-programme 2 qui lui-même peut appeler un sous-programme 3, etc... La seule limite tient à la capacité mémoire de votre ordinateur. Faites attention à ce qu'il y ait bien toujours une instruction RETURN qui corresponde à chaque instruction GOSUB.

4. Les sous-programmes peuvent être appelés également avec l'instruction ON X GOSUB.

Dans le prochain chapitre, nous allons nous attaquer à la structure et à la technique des menus.

4.3 LA TECHNIQUE DES MENUS

Maintenant que vous avez avancé dans la maîtrise de la programmation en BASIC, vous voudrez certainement un jour écrire vous-même des programmes d'une taille importante, soit pour vous faciliter une tâche donnée, soit pour commercialiser ces programmes. En tout cas, il sera important que vos programmes soient commodes et agréables à utiliser.

Il importe donc que l'utilisateur éventuel n'ait pas à chercher trop longtemps sur quelle touche il doit appuyer pour obtenir tel ou tel résultat ou fonction. Bien sûr un programme de taille importante doit être accompagné d'un manuel mais il ne faut pas non plus que l'utilisateur ait besoin de recourir sans cesse au manuel pour "deviner" comment il va pouvoir mettre en route telle ou telle opération.

Nous n'allons pas vous demander d'écrire un manuel mais nous voudrions simplement que vous maîtrisiez bien les principes de la technique des menus.

Comme nous l'avons déjà indiqué pour notre programme de cours de calcul, un menu est une "page" d'écran qui expose les différents points du programme, les différentes opérations que celui-ci permet. L'utilisateur peut alors sélectionner un de ces points du programme en appuyant sur la touche qui est associée dans le menu au point du programme qu'il a choisi. La présentation du menu est bien sûr laissée à l'imagination du programmeur mais il convient toutefois de toujours essayer de la rendre la plus claire possible. Il est toujours bien venu d'utiliser des séparations visuelles constituées de différentes suites de caractéres, pourvu que ces séparations ne surchargent pas l'écran.

Nous allons maintenant voir par un exemple concret comment on crée un menu. Nous allons essayer de créer un programme de tables mathématiques que l'utilisateur pourra consulter.

Supposons que nous ayons besoin des opérations suivantes :

racine carrée
sinus
cosinus
logarithme népérien
logarithme décimale

Notre programme doit donc pouvoir exécuter un de ces 5 calculs au choix.

Il manque encore un point à notre menu : la fin du programme, de façon à ce que l'utilisateur puisse mettre fin au programme dans le cadre de celui-ci, sans utiliser Control-C ni éteindre l'ordinateur. Notre menu offrira donc 6 options. Il faudra d'autre part que notre menu envoie un message à l'utilisateur pour l'inviter à entrer un chiffre ou une lettre comme par exemple :

Entrez votre choix: (1-6) ?

Il ne nous manque plus qu'un titre et un cadre pour améliorer la présentation du menu. Il serait bien que le titre du programme demeure à l'écran, quelle que soit la partie du programme exécutée. Il nous faudra donc utiliser un sous-programme pour le titre. Voici comment notre menu devra apparaître sur l'écran :

```
****************************************
*                                      *
  *TABLES MATHEMATIQUES*
*                                      *
****************************************

1 RACINE CARREE

2 SINUS

3 COSINUS

4 LOGARITHME NEPERIEN

5 LOGARITHME DE BASE 10

6 FIN DU PROGRAMME

ENTREZ VOTRE CHOIX: (1-6)
```

La dernière ligne doit en même temps recevoir l'entrée du chiffre choisi. Utilisons tout d'abord l'instruction INPUT pour l'entrée et nous verrons plus tard si nous pouvons employer ici l'instruction CHR$(INP(2)). Voici comment se présente le programme permettant de créer ce menu :

```
10    REM **********************
20    REM * DEBUT DU PROGRAMME *
30    REM **********************
40    REM
50    CLEARW 2:FULLW 2
60    DIM M$(6)
70    FOR I=1 TO 6
80    READ M$(I):NEXT I
90    DATA " 1 Racine carrée"
```

```
100  DATA " 2 Sinus"
110  DATA " 3 Cosinus"
120  DATA " 4 Logarithme naturel"
130  DATA " 5 Logarithme de base 10"
140  DATA " 6 Fin du programme"
150  GOTO 330
160  REM ********************
170  REM * SOUS-PROGRAMMES *
180  REM ********************
190  REM
200  REM ***************
210  REM * LIGNE-TITRE *
220  REM ***************
230  CLEARW 2
240  FOR I=1 TO 40:PRINT"*";:NEXT I:PRINT
250  PRINT"*                                    *";
260  PRINT"* TABLES MATHEMATIQUES *";
270  PRINT"*                                    *";
280  FOR I=1 TO 40:PRINT"*";:NEXT I:PRINT
290  RETURN
300  REM ********
310  REM * MENU *
320  REM ********
330  GOSUB 230
390  FOR I=1 TO 3:PRINT:NEXT I
400  FOR I=1 TO 6
410  PRINT M$(I)
420  NEXT I
430  PRINT
450  PRINT" Entrez votre choix (1-6)";
460  W$=CHR$(INP(2))
.
.
```

La ligne 50 vide l'écran et les lignes 60 à 80 créent le tableau M$ avec les données figurant dans les lignes DATA 90 à 140. Le programme saute alors par dessus les sous-programmes à la ligne 330.

Il saute alors au sous-programme en ligne 230 qui crée la ligne-titre. Les lignes 240 et 280 créent les lignes d'étoiles qui entoureront la ligne-titre.

Le programme quitte alors le sous-programme pour retourner en ligne 340. La ligne 390 sort 3 lignes vides pour que les points du menu n'effacent pas la ligne-titre. Les lignes 400 à 420 sortent alors les différents points du menu qui figurent dans le tableau. La ligne 450 demande enfin à l'utilisateur d'entrer une valeur. Comme l'instruction PRINT de cette ligne est suivie d'un point-virgule, l'entrée avec INPUT sera attendue immédiatement après la parenthèse (1-6).

Voilà donc comment nous avons créé notre menu. Il faudrait encore contrôler si la valeur entrée avec CHR$(INP(2)) est correcte.

Remarquons encore simplement que vous pouvez appeler le sous-programme de création de la ligne-titre de n'importe quel endroit de votre programme, avec l'instruction :

```
GOSUB 230
```

Notre but n'était pas ici de réaliser un programme complet mais uniquement de vous montrer la marche à suivre pour créer un menu. Cependant cela constituerait pour vous un excellent exercice si vous essayiez de compléter notre projet de programme. Le principe à suivre sera d'ailleurs le même que pour le programme de cours de calcul. Mais nous allons maintenant revenir plus en détail sur l'instruction CHR$(INP(2)).

4.3.1 UTILISATION DE ROUTINES CHR$(INP(2)) DANS LE MENU

Lorsque nous avons appris à utiliser l'instruction CHR$(INP(2)), vous avez peut-être vous-même remarqué que cette méthode est très "primitive" puisqu'elle ne permet pas à l'utilisateur de corriger

son entrée et qu'elle le contraint à entrer exactement le nombre de caractères attendu. C'est ainsi que dans notre dernière version du cours de calcul, l'utilisateur doit entrer 054 pour 54 car le programme attend de toute façon trois chiffres.

C'est pourquoi il convient de reconnaître que l'instruction INPUT est préférable et tout à fait satisfaisante pour un usage privé. Mais si vous voulez réaliser un programme de niveau commercial, il faut utiliser l'instruction CHR$(INP(2)) qui vous permet seule de protéger votre programme et l'utilisateur contre des erreurs de manipulation.

Nous allons donc développer une routine d'entrée avec l'instruction CHR$(INP(2)). Vous pourrez intégrer cette routine sous forme d'un sous-programme dans vos propres programmes. La première ligne en est maintenant connue :

```
10 A$=CHR$(INP(2))
```

Cette ligne vous permet de lire tous les caractères entrés à partir du clavier et de les affecter au fur et à mesure à A$. Dans notre application, nous n'autoriserons que les chiffres. Il nous faut donc sélectionner les caractères à entrer et exclure toutes les autres touches :

```
10 A$=CHR$(INP(2))
20 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 10
```

L'instruction IF ... THEN exclut ici tous les caractères dont le code ASCII n'est pas compris entre 48 et 57 qui sont les codes de 0 et 9. Si un caractère incorrect est entré, il est donc ignoré et le programme saute à nouveau à la ligne d'entrée.
Maintenant, si nous ne voulons autoriser que des nombres comprenant au maximum un certain nombre de chiffres, il nous faut compter le nombre de caractères corrects entrés :

```
10 A$=CHR$(INP(2))
20 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 10
30 Z=Z+1
40 IF Z > 4 THEN 10
```

Il nous a fallu introduire deux lignes supplémentaires : la ligne 30 qui augmente la valeur du compteur Z chaque fois qu'un caractère correct a été entré et la ligne 40 qui empêche que plus de 4 caractères soient pris en compte.

Il nous faut maintenant trouver un moyen pour indiquer à notre routine quand une entrée est terminée. Nous pouvons utiliser tout simplement pour ce faire la touche RETURN dont le code ASCII est 13. Il suffira de tester si ASC(A$) est égal à 13. Mais il faut faire attention à l'endroit où nous pourrons intégrer ce test. En effet si nous le plaçons après la ligne 20, la touche RETURN sera ignorée comme toutes les touches qui ne correspondent pas à des chiffres. Voici notre dispositif d'entrée complété :

```
10 A$=CHR$(INP(2))
15 IF ASC(A$) = 13 THEN 100
20 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 10
30 Z=Z+1
40 IF Z > 4 THEN 10
```

Il nous faut maintenant stocker dans une variable les différents caractères qui pourront être entrés dans ce dispositif. Il faut également que les caractères entrés soient sortis sur l'écran :

```
10 A$=CHR$(INP(2))
15 IF ASC(A$) = 13 THEN 100
20 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 10
30 Z=Z+1
40 IF Z > 4 THEN 10
50 B$=B$+A$
60 PRINT A$;
```

La ligne 60 sort les caractères à partir de l'emplacement actuel du curseur. Les différents caractères seront sortis l'un à la suite de l'autre grâce au point-virgule. Mais il faut ensuite que le programme retourne en ligne 10 après avoir sorti chaque caractère :

```
10 A$=CHR$(INP(2))
15 IF ASC(A$) = 13 THEN 100
20 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 10
30 Z=Z+1
40 IF Z > 4 THEN 10
50 B$=B$+A$
60 PRINT A$;
70 GOTO 10
```

Il ne nous reste plus qu'à convertir la chaîne de caractères B$ en une valeur numérique et à affecter celle-ci à une variable numérique. Ceci doit être fait une fois que l'utilisateur a actionné la touche RETURN pour mettre fin à son entrée. Comme nous voulons d'autre part utiliser cette routine sous la forme d'un sous-programme, nous ne devons pas oublier de réinitialiser à 0 le compteur Z chaque fois que la touche RETURN a été actionnée. Sinon l'utilisateur ne pourrait, lors du prochain appel de cette routine, entrer qu'un nombre de chiffres égal à 4 moins la valeur atteinte par Z lors de sa première utilisation.

Nous pourrions enfin prévoir de sortir en fin de routine la valeur numérique entrée par l'utilisateur, puis il serait temps de terminer notre routine par une instruction RETURN :

```
10 A$=CHR$(INP(2))
15 IF ASC(A$) = 13 THEN 100
20 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 10
30 Z=Z+1
40 IF Z > 4 THEN 10
50 B$=B$+A$
60 PRINT A$;
70 GOTO 10
```

```
100 B=VAL(B$)
110 PRINT B
120 END
```

Nous avons ainsi réalisé une routine avec CHR$(INP(2)) qui nous permet d'entrer et de sortir sur l'écran des nombres comportant jusqu'à 4 chiffres. Si vous voulez pouvoir entrer des nombres plus importants, il vous suffit de changer la valeur comparée à Z en ligne 40.

Cette routine est plus pratique que celle que nous avions dans notre programme de cours de calcul. Il lui manque cependant une fonction importante, qui est d'ailleurs assez complexe à réaliser, à savoir la possibilité d'effacer les valeurs qu'on vient d'entrer. Voici une routine qui possède cette fonction :

```
10  REMROUTINECHR$(INP(2))FONCTIONSUPPRESSION
20  A$=CHR$(INP(2))
30  IF ASC(A$) = 13 THEN 130
40  IF ASC(A$) <> 8 THEN 70
50  IF LEN(B$) < 1 THEN 20
60  B$=LEFT$(B$,LEN(B$)-1):Z=Z-1:GOTO 110
70  IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 20
80  Z=Z+1
90  IF Z > 4 THEN Z=4:GOTO 20
100 B$=B$+A$
110 PRINT A$;
120 GOTO 20
130 B=VAL(B$):Z=0
140 PRINT B
150 END
```

Voyons les lignes que nous avons ajoutées à la routine précédente. Tout d'abord la ligne 40 qui teste si la touche BACKSPACE a été actionnée. La valeur ASCII de cette touche est en effet 8. Si cette touche n'a pas été actionnée, le programme saute à la ligne 70. Si par contre la touche BACKSPACE a été actionnée,

la ligne 50 teste si la variable alphanumérique B$ contient encore au moins un caractère. Si donc la variable B$ est vide, la touche BACKSPACE sera ignorée et le programme sautera à nouveau à la ligne 20.

L'effaçage d'un caractère est réalisé par la ligne 60 :

```
B$=LEFT$(B$,LEN(B$)-1)
```

qui raccourcit d'un caractère la variable B$. En effet l'instruction LEFT$ génère ici une chaîne de caractères comprenant les X caractères les plus à gauche de B$ mais X est ici remplacé par LEN(B$)-1, c'est-à-dire la longueur antérieure de B$ moins 1. La nouvelle chaîne ainsi générée est à nouveau affectée à B$. Ce procédé est semblable à l'instruction très courante qui permet de diminuer la valeur d'une variable numérique :

```
A=A-1
```

La ligne 60 a donc pour effet de supprimer le caractère le plus à droite de B$, c'est-à-dire aussi le dernier entré. Le compteur du nombre de chiffres entrés Z est alors tout naturellement diminué de 1 de façon à ce que le caractère qui a été supprimé ne soit pas pris en compte par le test de la ligne 90 qui contrôle que l'utilisateur n'a pas entré plus de 4 chiffres.

Le caractère qui vient donc d'être supprimé de la variable qui mémorise votre entrée ainsi que de la variable-compteur Z qui compte combien de chiffres vous avez entrés, ce caractère se trouve cependant toujours sur l'écran. C'est pourquoi la dernière instruction de la ligne 60 est une instruction de saut en ligne 110 où la valeur actuelle de A$ est sortie sur l'écran par l'instruction PRINT A$;. Lorsque vous venez d'appuyer sur la touche BACKSPACE, A$ contient le code ASCII 8.
Ce code entraîne, sorti avec PRINT, un retour en arrière du curseur d'une position. Entrez par exemple :

PRINT"Atari SX";CHR$(8);"T"

Vous voyez que vous obtenez à l'écran la sortie suivante :

Atari ST

En effet le code de commande 8 raméne le curseur d'une position en arrière. Le "T" remplace alors immédiatement le caractère qui vient d'être ainsi supprimé.

Nous avons ainsi développé une routine CHR$(INP(2)) qui est presque aussi simple à manier pour l'utilisateur qu'une routine INPUT mais qui présente en outre l'avantage de nous permettre de sélectionner les touches autorisées pour entrer une donnée.

Vous pouvez maintenant rendre notre routine plus générale en autorisant, outre les chiffres, les lettres et les espaces. Il vous faut ajouter pour cela deux instructions IF ... THEN supplémentaires pour interdire les caractères dont les valeurs ASCII (de 58 à 64) sont entre celles des chiffres et celles des lettres. Maintenant que vous connaissez le principe de la sélection des touches, vous pourrez donc déterminer vous-même, pour chaque application, quelles touches vous souhaitez autoriser ou éliminer.

Intéressons-nous maintenant au positionnement du curseur, grâce à l'instruction **GOTOXY**

4.3.2 POSITIONNEMENT DU CURSEUR AVEC GOTOXY

GOTOXY permet de positionner le curseur dans un emplacement déterminé de l'écran.

ATTENTION ! Dans la version actuelle du BASIC, au moment où ce livre a été écrit, cette instruction ne fonctionne pas toujours correctement.

La syntaxe de GOTOXY est :

GOTOXY X,Y

Les paramètres sont ici:

X = numéro de colonne (0-39)

Y = numéro de ligne (0-24)

Si vous sortez un texte ensuite avec PRINT, il sera en général placé suivant l'instruction de positionnement. L'exemple suivant le montre :

```
10 CLEARW 2
20 GOTOXY 19,12
30 PRINT "SORTIE AVEC GOTOXY!"
40 END
```

Nous connaissons ainsi une possibilité très pratique pour effectuer des sorties de texte en un endroit quelconque de l'écran.

4.4 PROCEDURES DE TRI

Le tri de données, qu'il s'agisse de données numériques ou de textes est une des tâches les plus classiques des programmeurs. Le critère de tri est naturellement la grandeur même des nombres lorsqu'il s'agit de données numériques et l'ordre alphabétique lorsqu'il s'agit de textes.

Pour trier des données on peut faire appel à quantité d'algorithmes bien connus qui se différencient les uns des autres d'une part par leur complexité et d'autre part par leur efficacité. Pour choisir le bon algorithme pour une application donnée il convient tout d'abord de tenir compte du nombre de données qu'il s'agit de trier.

Si vous n'avez en effet qu'une dizaine ou une cinquantaine de données à trier, les procédures les plus simples suffiront en général aisément à la tâche. Par contre si la masse de données excède une centaine ou un millier, les procédures les plus simples se révèleront le plus souvent impraticables car le temps de tri risque de dépasser plusieurs heures. Nous allons vous présenter la méthode de tri la plus simple, qui est appelée bubble sort.

Avec bubble sort on compare deux éléments voisins qu'on échange si nécessaire. Une fois que tous les éléments d'un fichier ou d'un tableau ont été ainsi comparés, ils se trouvent dans l'ordre voulu.

Nous allons d'abord créer un tableau de 6 éléments que nous remplirons de valeurs aléatoires :

```
10  REM CREATION D'UN TABLEAU
20  DIM F(6)
30  FOR I=1 TO 6
40  A=INT(50*RND(0))+1
50  F(I)=A
60  NEXT I
```

Nous allons maintenant écrire notre procédure de tri. Bien sûr nous pourrions la réaliser en employant une boucle FOR ... NEXT mais nous pensons que l'algorithme utilisé sera plus évident si nous utilisons des instructions IF ... THEN. Une fois que vous aurez bien compris l'algorithme employé, vous pourrez bien sûr fort bien le mettre en oeuvre avec une boucle FOR ... NEXT. Voici donc notre routine de tri :

```
.
.
100  REM BUBBLE-SORT
110  Z=0
120  IF F(6) > F(5) THEN 140
130  F(0)=F(6):F(6)=F(5):F(5)=F(0):Z=1
140  IF F(5) > F(4) THEN 160
```

```
150  F(0)=F(5):F(5)=F(4):F(4)=F(0):Z=1
160  IF F(4) > F(3) THEN 180
170  F(0)=F(4):F(4)=F(3):F(3)=F(0):Z=1
180  IF F(3) > F(2) THEN 200
190  F(0)=F(3):F(3)=F(2):F(2)=F(0):Z=1
200  IF F(2) > F(1) THEN 220
210  F(0)=F(2):F(2)=F(1):F(1)=F(0):Z=1
220  IF Z=1 THEN 110
230  FOR I=1 TO 6
240  PRINT F(I)
250  NEXT I
260  END
```

Z est tout d'abord initialisé à 0 en ligne 110. Nous verrons plus loin pourquoi. La ligne 120 effectue la première comparaison : si F(6) est déjà supérieur à F(5), aucun échange n'est nécessaire et le programme saute directement à la seconde comparaison en ligne 140. Si par contre F(6) est inférieur à F(5), la ligne 130 échange entre eux les contenus de ces deux éléments. C'est ici l'élément F(0) que nous utilisons comme variable de stockage provisoire pendant l'échange. Le même principe a été appliqué dans les lignes suivantes du programme.

Après chaque échange, la variable-compteur Z reçoit la valeur 1 alors que Z valait 0 en début de programme. Nous pouvons donc savoir grâce à la valeur de Z si au moins un échange de valeurs a été nécessaire: si Z vaut 0, cela signifie que les valeurs à trier étaient dans l'ordre voulu, si Z vaut 1 cela signifie qu'au moins deux valeurs n'étaient pas dans l'ordre voulu. Ce procédé qui consiste à utiliser une variable comme témoin d'une situation est très courant en informatique. Les variables-témoins sont appelées FLAGS, ce qui veut dire drapeaux. L'intérêt est ici que si aucune opération d'échange n'a été effectuée, donc si Z vaut 0, le programme sait qu'il peut mettre fin au tri.

Les dernières lignes de notre routine sortent le tableau trié sur l'écran. Mais si vous voulez observer le déroulement du tri, vous pouvez modifier ainsi les dernières lignes du programme :

```
220 FOR I=1 TO 6
240 PRINT F(I);
250 NEXT I
260 IF Z=1 THEN 110
270 END
```

Nous vous invitons maintenant à bien étudier cet algorithme de tri pour que vous n'ayez pas de mal à en aborder plus tard de plus complexes mais aussi plus performants. Cet algorithme peut être utilisé sans problème pour trier un tableau d'environ 100 éléments.

CHAPITRE 5

LE TRAVAIL AVEC LE LECTEUR DE DISQUETTE

5.1 LA GESTION DES PROGRAMMES

Il n'est pas un possesseur d'ordinateur qui s'imagine, s'il éteint son ordinateur, qu'il ait une chance de retrouver les lignes de programme qu'il avait mis tant de temps à entrer lorsqu'il allumera à nouveau son ordinateur. Evidemment : la mémoire interne de l'Atari ST est une mémoire dynamique, c'est-à-dire qu'elle doit être en permanence rafraîchie, ce qui n'est possible que si l'ordinateur est en état de marche, pour que les bits et octets qui la composent ne s'évanouissent pas. C'est pour cela et uniquement pour cela qu'il faut absolument archiver vos programmes sur un support externe de données. L'époque où l'on s'accomodait, pour cette tâche, du lecteur de cassette, est révolue. Atari offre le lecteur de disquette SF354 pour un prix tout à fait acceptable.

Sur ce lecteur de disquette, vous pouvez placer 360 kilo-octets de données, sous la forme de programmes ou d'autres types de fichiers. Comme pour beaucoup d'autres instructions, la liste présentée par le menu du BASIC contient toutes les instructions qui sont nécessaire pour la sauvegarde de programmes. Ce mode d'emploi vraiment enfantin ne nécessite certainement pas d'autres explications. Nous allons donc, dans cette section, nous intéresser aux instructions qui concernent la gestion des programmes.

5.1.1 SAUVEGARDE DE PROGRAMMES

Sur l'Atari ST, il y a deux possibilités de sauvegarder un programme : la sauvegarde avec suppression et la sauvegarde sans suppression.

L'instruction **SAVE** sauvegarde un programme sous un nom sous lequel ne figure encore aucun autre programme sur la disquette :

SAVE **nom, intervalle de lignes**

Le paramètre nom définit le nom du programme. Ce nom est ainsi structuré :

xxxxxxxx.yyy

Le nom de fichier proprement dit comporte au maximum 8 caractères. Le suffixe du nom est indiqué à la suite d'un point et il comporte au maximum 3 caractères. Ce suffixe permet de classifier les fichiers. Si aucun suffixe (extension) n'est fourni, le suffixe .BAS est automatiquement ajouté.

Le paramètre intervalle de lignes correspond à celui des instructions LIST ou DELETE.
Si aucun intervalle de lignes n'est fourni, le programme est stocké entièrement.

Cette instruction entraîne un message d'erreur s'il existe déjà sur la disquette un programme portant le nom que vous avez indiqué.

Notez que le nom de programme ne doit pas être écrit entre guillemets.

Voici maintenant la deuxième possibilité de sauvegarder un programme :

REPLACE nom, intervalle de lignes

Cette instruction permet de remplacer un programme figurant déjà sur la disquette par une version plus récente de ce programme. Les paramètres suivent les mêmes règles que pour l'instruction SAVE.

5.1.2 CHARGEMENT DE PROGRAMMES

Le BASIC de l'Atari ST fournit également deux instructions pour le chargement des programmes mais ces deux instructions fonctionne de façon tout à fait identique.

OLD nom

LOAD nom

Si aucune extension n'est indiquée, c'est le programme portant l'extension .BAS qui sera chargé.
Si le programme indiqué n'est pas trouvé, cela vous sera indiqué par le message d'erreur :

File not found on disk drive specified

5.1.3 AFFICHAGE DU CONTENU DE LA DISQUETTE

L'instruction **DIR** permet de faire afficher le contenu d'une disquette :

DIR lecteur:masque

Si l'instruction est entrée sans masque, c'est le contenu intégral de la disquette qui sera affiché. Le masque vous permet de faire afficher des groupes de fichiers déterminés. Exemples :

DIR A:*.BAS affichera tous les programmes BASIC du lecteur A

DIR B:TEST.* affichera tous les fichiers appelés TEST, quelle que soit leur extension

DIR A:A*.B* affichera les fichiers dont le nom commence par A et dont l'extension commence par B.

L'étoile signifie donc que tous les caractères à partir de la position de l'étoile doivent être ignorés.

Le point d'interrogation joue également un rôle important dans l'instruction DIR. Il permet de marquer tous les emplacements du masque qui ne doivent pas être pris en compte dans la sélection des fichiers à afficher. Le masque ?FILE.BAS permettrait, par exemple, de faire afficher les fichiers :
AFILE.BAS, BFILE.BAS, CFILE.BAS etc...

5.1.4 SUPPRESSION DE FICHIERS

Si une disquette est tellement pleine que vous ne pouvez plus y stocker aucun programme, il faut la nettoyer. Il y a souvent sur une disquette quelques anciennes versions de programmes dont vous n'avez plus besoin depuis longtemps.

Utilisez pour les supprimer l'instruction :

ERA lecteur:nom

L'instruction ERA A:TEST.BAS supprimera par exemple le programme TEST.BAS sur la disquette figurant dans le lecteur A. Au lieu du nom de fichier, vous pouvez également fournir un masque comme celui qui a été décrit pour l'instruction DIR.

ERA A:*.*

supprimera par exemple tous les fichiers sur le lecteur A.

Nous savons déjà que le BASIC de notre Atari ST contient plusieurs instructions qui sont pratiquement identiques les unes aux autres. Pour cette instruction également, il existe un équivalent :

KILL "lecteur:nom"

La seule différence semble être constituée par les guillemets dans lesquels doit être indiqué le nom de fichier. Cela présente un avantage non-négligeable: le paramètre est constitué par une chaîne de caractères. Il est par conséquent possible, avec l'instruction KILL, d'utiliser la séquence d'instructions suivante :

```
B$="PROG1.BAS"
KILL B$
```

L'avantage de cette suppression "paramétrée" est que vous pouvez entrer au clavier un nom de fichier, à l'intérieur d'un programme. Le programme peut ensuite se charger de la suppression du fichier. Cela ne marcherait pas avec l'instruction ERA.

5.1.5 CHANGEMENT DU NOM DE FICHIERS

Comme tout bon BASIC, celui de l'Atari ST vous donne aussi la possibilité d'exécuter des opérations DOS. Le changement du nom d'un fichier fait partie des fonctions DOS élémentaires.

L'instruction BASIC correspondante est :

NAME ancien fichier AS nouveau fichier

Par exemple, si le programme TEXT05.BAS doit être rebaptisé en TEXTPRO.BAS, vous entrerez :

NAME TEXT05.BAS AS TEXTPRO.BAS

5.2 GESTION DE FICHIER SEQUENTIEL

Dans cet ouvrage, vous avez déjà eu l'occasion de traiter des données telles que des noms par exemple. Ces données faisaient cependant toujours partie intégrante du programme, dans des DATAs. Il est évident que ce n'est pas là une méthode habituelle.

On ne peut pas attendre de l'utilisateur d'un programme qu'il doive, pour entrer ou modifier des données, effectuer une modification du programme. Dans d'autres langages de programmation, il serait même nécessaire de réaliser une version complètement nouvelle du programme. Aberrant !

Le principe de la méthode correcte vous apparaît certainement : les données doivent être gérées sous la forme d'un fichier indépendant sur la disquette. Nous allons traiter dans ce chapitre de la forme d'organisation des données la plus connue.

Dans un fichier séquentiel, les enregistrements sont rangés à la suite les uns des autres, séparés entre eux par le caractère ASCII RETURN - CHR$(13) -.
Ce RETURN est nécessaire car l'instruction de lecture d'un enregistrement d'un fichier séquentiel lira l'enregistrement jusqu'à ce qu'elle rencontre ce RETURN.

Nous traiterons de la gestion de fichier séquentiel à travers quelques sous-programmes d'exemple.

Supposons que nous voulions placer sur la disquette une liste d'adresse figurant dans l'ordinateur sous la forme d'un tableau à deux dimensions. Chaque enregistrement d'adresse se compose de trois champs :

A$(n,1) Nom
A$(n,2) Prénom
A$(n,3) Numéro de téléphone

La liste totale doit compter 200 entrées. Dans une telle gestion de fichier, il est toujours indispensable de gérer un compteur d'enregistrements qui indiquera en permanence le numéro du dernier enregistrement du tableau. Nous l'appellerons COMPTEUR.
Ce compteur est par exemple nécessaire lorsqu'on veut entrer un nouveau nom car il permet de déterminer le prochain emplacement libre du tableau (COMPTEUR+1).

Pour sortir ce tableau d'adresses sur l'écran, nous pourrions utiliser un sous-programme qui se présenterait à peu près ainsi :

```
1000  REM AFFICHER ADRESSES
1010  :LISTADDR
1020  GOSUB MASQUE ECRAN
1030  FOR X=1 TO COMPTEUR
1040  PRINT"NOM:        ";A$(X,1)
1050  PRINT"PRENOM:     ";A$(X,2)
1060  PRINT"TELEPHONE: ";A$(X,3)
1070  PRINT
1080  PRINT"RETURN POUR CONTINUER"
1090  IF INP(2)<>13 THEN 1090
1100  NEXT X
1110  RETURN
```

Ce sous-programme est facile à "lire". Nous ne l'expliquerons donc pas plus en détail. L'important est maintenant de savoir comment nous allons pouvoir écrire ces enregistrements sur la disquette. Le sous-programme de sauvegarde des enregistrements doit d'abord ouvrir un fichier dans lequel les adresses seront écrites. L'instruction d'ouverture d'un fichier séquentiel en écriture est la suivante :

OPEN "O",#numéro de fichier,"nom de fichier"

Le fichier d'adresses devra s'appeler ADRESSES.DAT :

OPEN "O",#1,"ADRESSES.DAT"

Le numéro de fichier est nécessaire pour distinguer entre eux plusieurs fichiers ouverts.

Voici maintenant le sous-programme d'écriture des adresses :

```
2000  REM SAUVEGARDER ADRESSES
2010  :SAVEADDR
2020  OPEN "O",#1,"ADRESSES.DAT"
2025  PRINT #1,COMPTEUR
2030  FOR X=1 TO COMPTEUR
2040  FOR Y=1 TO 3
2050  PRINT #1,A$(X,Y)
2060  NEXT Y,X
2070  CLOSE #1
2080  PRINT "ADRESSES SAUVEGARDEES!"
2090  PRINT "RETURN POUR CONTINUER"
2100  IF INP(2)<>13 THEN 2100
2110  RETURN
```

Après l'ouverture du fichier, le compteur d'enregistrements est sauvegardé en premier. Il sera plus tard nécessaire pour le chargement sélectif des enregistrements d'adresse. Les différents champs du tableau sont sauvegardés à travers deux boucles imbriquées. On sauvegarde donc tout d'abord les nom, prénom, numéro de téléphone du deuxième enregistrement, etc... Pour écrire dans un fichier séquentiel, on utilise l'instruction PRINT avec le numéro de fichier comme suffixe. Voici maintenant le sous-programme pour charger ce fichier :

```
3000  REM CHARGER ADRESSES
3010  :LOADADDR
3020  OPEN "I",#1,"ADRESSES.DAT"
3025  INPUT #1,COMPTEUR
3030  FOR X=1 TO COMPTEUR
3040  FOR Y=1 TO 3
3050  INPUT #1,A$(X,Y)
3060  NEXT Y,X
3070  CLOSE #1
3080  INPUT "FICHIER CHARGE!"
3090  INPUT "RETURN POUR CONTINUER"
3100  IF INP(2)<>13 THEN 3100
```

```
3110 RETURN
```

Le mode de l'instruction OPEN est, pour la lecture d'un fichier, I (Input). Après que le compteur d'enregistrements ait été lu, ce mode est utilisé comme argument pour la boucle extérieure. Le sous-programme sait donc, grâce à ce compteur d'enregistrements, jusqu'où il doit lire.

A l'aide de ces sous-programmes que vous pouvez aisément adapter à vos besoins personnels, vous êtes maintenant en mesure de créer vos propres gestions de fichier.

CHAPITRE 6

MUSIQUE ET GRAPHISME

Un ordinateur se prête parfaitement à la production de sons. Le processeur, qui est le véritable moteur de l'ordinateur, est commandé par un générateur de fréquence. Ce générateur produit selon l'ordinateur environ 1 à 8 MHz (millions de battements par seconde). Le moyen le plus simple consiste à placer cette fréquence sur un haut-parleur. Comme les sons ainsi générés ne peuvent être perçus par l'oreille humaine, la fréquence doit être beaucoup plus basse. Cela est obtenu en divisant la fréquence d'horloge avant de la sortir sur le haut-parleur. Les fréquences ainsi générées seront alors perçues comme autant de notes.

C'était la première génération de la production de sons sur l'ordinateur. Comme ces sons, comparables à l'alarme d'une montre bracelet, n'étaient pas satisfaisants, la production des sons sur les ordinateur familiaux se développa sans cesse. Le résultat est que de nombreux ordinateurs familiaux sont dotés de ce qu'on appelle des synthétiseurs qui sont logés sur un circuit intégré. Les sons ainsi produits ressemblent à s'y méprendre à ceux des instruments "naturels".

Le circuit intégré produisant les sons sur l'Atari ST ne peut pas être qualifié de synthétiseur à proprement parler. Il lui manque des propriétés importantes telles que plusieurs formes d'onde différentes (sinus, rectangle, etc.) ainsi que des filtres permettant d'influer sur la sonorité. Le principe du synthétiseur qui consiste à faire dépendre la fréquence (hauteur) des sons d'une tension donnée n'est pas non plus réalisé.

Le générateur de son de l'Atari ST dispose de trois voix. On peut donc faire entendre simultanément trois sons différents.

Tout ce qui est produit par le générateur de son retentit à travers le haut-parleur intégré dans le moniteur. Le volume du haut-parleur intégré peut être réglé sur le moniteur.

Le chapitre suivant vous présente l'instruction de production des sons. Nous ne pouvons toutefois décrire ici en détail tous les problèmes posés par la production des sons.

SOUND

Le générateur de sons de l'Atari ST est commandé par l'instruction **SOUND** :

SOUND canal,volume,note,octave,durée

Signification des paramètres :

canal : canal sonore (1 à 3)

volume : de 0 à 15 (0=muet, 15=fort)

note : de 1 à 12 (1=do, 2=do dièse, ... 12=si)

octave : de 1 à 8 (1=bas, 8=haut)

durée : durée de la note en 50èmes de seconde

Voici d'abord quelques notions fondamentales de musique. Ce qu'on appelle la gamme (chromatique) se compose de 12 notes comme vous pouvez le voir si vous examinez le clavier d'un piano.

Paramètre de note	Note de la gamme chromatique
1	DO
2	DO DIESE
3	RE
4	RE DIESE
5	MI
6	FA
7	FA DIESE
8	SOL
9	SOL DIESE
10	LA
11	LA DIESE
12	SI

Un clavier se compose de plusieurs fois cette section. Ces suites de notes sont appelées octaves. La première octave comprend les notes basses, la dernière octave les notes hautes.

Le programme suivant, "ST-ORGEL", transforme votre Atari ST en un instrument de musique.

```
10 REM ST-ORGEL
20 DIM TON(256)
30 TON$="Q2W3ER5T6Y7UI9O0P"
40 FOR I=1 TO 17
50 INDEX=ASC(MID$(TON$,I,1))
60 TON(INDEX)=I
70 NEXT I
80 CLEARW 2:FULLW 2
90 PRINT"        ST-ORGEL"
100 PRINT
110 PRINT" 2 3  5 6 7  9 0"
120 PRINT
130 PRINT"Q W E R T Y U I O P"
```

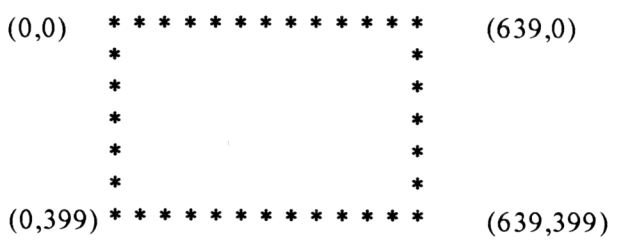
```
140 A=INP(2)
150 TON=TON(A)
160 OKT=3
170 IF TON>12 THEN TON=TON-12:OKT=4
180 SOUND 1,15,TON,OKT,20
190 SOUND 1,0,0,0,0
200 GOTO 140
```

Un petit programme pour un grand effet ! Après en avoir étudié le mode de fonctionnement, vous pourrez peut-être lui apporter des améliorations.

6.2 LE GRAPHISME

Pour beaucoup d'acquéreurs d'un des nombreux ordinateurs familiaux offerts sur le marché, la représentation des couleurs et surtout les possibilités graphiques constituent un critère d'achat décisif. L'Atari ST offre, dans sa classe de prix, une splendeur de couleurs incomparable et des propriétés graphiques excellentes. Le possesseur d'un Atari ST dispose ainsi d'innombrables couleurs qu'il peut mélanger lui-même à partir des trois couleurs fondamentales rouge, vert et bleu. La résolution graphique atteint 640 points sur 400 ce qui est vraiment extraordinaire pour un ordinateur de cette classe de prix.

Comme vous le savez certainement déjà, vous pouvez choisir entre trois résolutions graphiques différentes auxquelles correspondent des nombres de couleurs différents. Nous vous présenterons les instructions qui permettent de dessiner des figures uniquement en prenant l'exemple de la haute résolution. La haute résolution comporte 640 points image sur 400 et les couleurs noir et blanc. Pour que chaque point puisse être appelé, il faut se représenter l'écran comme un système de coordonnées. Un emplacement est déterminé, dans les instructions graphiques, de façon semblable à l'instruction GOTOXY qui permet de placer le curseur dans un endroit quelconque de l'écran: ici aussi, il faut fournir deux valeurs. La première valeur détermine la position sur l'axe horizontal (axe des X) de l'écran, en partant de la gauche. La seconde valeur correspond à la position sur l'axe vertical (axe des Y), en partant du haut. Les quatre angles de l'écran graphique ont donc les coordonnées suivantes :



Une fois que vous vous serez familiarisé avec ce système, nous pourrons aborder les premières instructions graphiques.

6.2.1 LES LIGNES

LINEF dex,dey,àx,ày

Une ligne allant de l'angle supérieur gauche à l'angle inférieur droit de l'écran sera donc produite par l'instruction :

LINEF 0,0,639,399

Il est très intéressant de marier le dessin de lignes avec des boucles :

```
10  CLEARW 2:FULLW 2
20  FOR i=0 TO 399 STEP 10
30   LINEF 399,i,0,399-i
40  NEXT i
```

Un programme simple mais spectaculaire. C'était trop beau, en voici un autre :

```
10  CLEARW 2:FULLW 2
20  FOR i=0 TO 399 STEP 10
30  LINEF I,0,399,I
40  LINEF 399,I,399-I,399
50  LINEF 399-I,399,0,399-I
60  LINEF 0,399-I,I,0
70  NEXT i
```

N'est-il pas admirable de voir apparaître de telles images à l'écran, surtout lorsqu'elles viennent de votre propre plume ? Vous pourriez peut-être vous essayer vous aussi à de telles phantaisie graphiques. C'est très amusant, même si ça ne marche pas toujours du premier coup.

Entre nous soit dit, j'ai aussi passé pas mal de temps avant de mettre au point cette image. La difficulté consiste à convertir les images telles qu'on les imagine dans le système de coordonnées, c'est-à-dire à trouver l'algorithme. Cela tient presque du sport cérébral.

6.2.2 CERCLES

Pour dessiner un cercle, il faut indiquer le centre du cercle ainsi que son rayon. Il est également possible de produire des sections de cercle en indiquant l'angle de départ et l'angle de fin. Cette instruction a la syntaxe suivante :

CIRCLE centrex,centrey,rayon,angledébut,anglefin

Les angles doivent être indiqués en dixièmes de dégré (0 à 3600).

L'exemple suivant dessine un cercle d'un rayon de 50 points au milieu de l'écran :

CIRCLE 320,200,50

Voici maintenant un programme qui illustre la production de sections de cercle :

```
10  FOR I=10 TO 100 STEP 5
20  CIRCLE 320,200,I,0,900
30  NEXT I
```

6.2.3 ELLIPSES

L'instruction de dessin des ellipses ne diffère de CIRCLE que par la distinction entre rayons X et Y.

ELLIPSE centrex,centrey,rayonx,rayony,angledébut,anglefin

L'exemple suivant dessine une ellipse d'un rayon x de 100 points et d'un rayon y de 50 points au milieu de l'écran :

```
ELLIPSE 320,200,100,50
```

Voici maintenant un programme de démonstration :

```
10  FOR X=50 TO 200 STEP 10
20  FOR Y=50 TO 125 STEP 5
30  ELLIPSE 320,200,X,Y
40  NEXT Y,X
```

6.2.4 SURFACES REMPLIES

Les cercles et les ellipses peuvent aussi être remplis de couleurs ou de hachures avec les instructions :

PCIRCLE

et

PELLIPSE

Les paramètres sont les mêmes que pour les instructions déjà décrites. Avant que les instructions de remplissage puissent être exécutées, certains attributs doivent être fixés avec l'instruction **COLOR** :

COLOR couleurtexte,couleurremplissage,couleurligne,style,index

Signification des paramètres :

couleurtexte	couleur de la sortie de texte
couleurremplissage	couleur du modèle de remplissage
couleurligne	couleur des lignes, cercles et ellipses
style	modèle de remplissage
index	type de remplissage

Les trois premiers paramètres seront toujours 1 (noir) ou 0 (blanc) sur un moniteur monochrome. Il est donc possible de supprimer les figures qui ont été dessinées en les redessinant dans la couleur du fond de l'écran. Pour dessiner, la couleur de ligne doit être fixée sur noir (COLOR 1,1,1), pour supprimer un dessin, elle doit être fixée sur blanc (COLOR 1,1,0).

Les modèles de remplissage sont déterminés avec les paramètres style et index. Le programme suivant illustre les 24 modèles de grille différents :

```
10  CLEARW 2:FULLW 2
20  FOR I=1 TO 24
30  COLOR 1,1,1,I,2
40  PCIRCLE 25*(I-1)+15,100,20
50  NEXT I
```

Voici maintenant les 12 modèles de hachures différents :

```
10  CLEARW 2:FULLW 2
20  FOR I=1 TO 12
30  COLOR 1,1,1,I,3
40  PCIRCLE 25*(I-1)+15,100,20
50  NEXT I
```

Indépendamment du paramètre style, le type de remplissage 0 produit des surfaces remplies de blanc et le type de remplissage 1 des surfaces remplies de noir. Le type de remplissage 4 met à votre disposition le logo d'Atari comme modèle de remplissage.

Ce chapitre avait pour but de vous inciter à faire vous-même des expériences graphiques intéressantes. Nous espérons vous avoir suffisamment inspiré.

CHAPITRE 7

FONCTIONS GEM SOUS BASIC

7.1. LES BASES DE GEM

Avant que vous n'utilisiez les fonctions GEM, il serait souhaitable que vous compreniez bien le principe du système d'exploitation GEM. GEM se compose essentiellement de deux parties :

VDI (Virtual Device Interface)

AES (Application Environment System)

AES et VDI ne sont rien d'autres que de très grandes collections de sous-programmes, de fonctions puissantes autant que pratiques. Ces fonctions sont intégrées dans les applications essentiellement à partir de l'assembleur ou du langage C pour être reliées lors de la compilation ou de l'assemblage. Il n'est donc pas recommandé d'écrire des applications GEM complètes en BASIC mais plusieurs fonctions peuvent parfaitement être utilement employées en BASIC.

VDI - GEM

La VDI contient toutes les fonctions graphiques de base comme le dessin de lignes, cercles, etc. Les fonctions qui déterminent l'image écran typique de GEM, avec les fenêtres, les boîtes etc. figurent dans AES. Si vous utilisez par exemple la fonction d'ouverture d'une fenêtre, c'est AES qui en est responsable. AES est à son tour aidé par les routines VDI qui produisent par exemple le cadre d'une fenêtre. VDI utilise à son tour les fonctions du DOS. La hiérarchie de GEM se présente donc ainsi :

DOS (Disk Operating System)

VDI (Virtual Device Interface)

AES (Application Environment System)

La tâche de VDI consiste à faciliter les applications graphiques grâce à ses fonctions graphiques. Le plus beau c'est que les fonctions sont indépendantes de la machine de sortie graphique. Les interfaces machine, qui font également partie de la VDI, se chargent de la commande des aspects propres à la machine.

AES - GEM

Pour expliquer ce qu'est AES, le mieux est de citer son nom : Application Environment System (système (de gestion) de l'environnement pour des programmes utilisateur). Cet "environnement" est dans GEM orienté graphisme, c'est-à-dire que la communication entre l'ordinateur et l'utilisateur s'effectue essentiellement à travers des éléments graphiques (par exemple les fenêtres et les pictogrammes).

L'environnement graphique de AES est un système d'exploitation particulièrement puissant qui se compose de plusieurs éléments :

Les bibliothèques de routines offrent des fonctions qui permettent de contrôler et d'utiliser tous les éléments du système AES. Le "multi-tâches" permet à plusieurs programmes d' être exécutés apparemment simultanément. Le "shell" établit la relation avec le système d'exploitation proprement dit, le TOS. Pour que la gestion des éléments de l'écran soit possible, AES gères des buffers (mémoires provisoires) qui servent au stockage intermédiaire de sections de l'écran.

7.2 TRANSMISSION DE PARAMETRES AUX ROUTINES GEM

Les routines GEM sont tellement complexes qu'elles nécessitent non pas seulement différentes variables mais des tableaux entiers de paramètres d'entrée et de sortie. Par exemple la fonction VDI d'ouverture d'un périphérique de travail ne comprend par exemple pas moins de 14 paramètres en entrée et 60 en sortie !

Mais ne craignez rien, nous n'allons vous présenter que quelques-unes des très nombreuses fonctions GEM (1). Expliquons d'abord le principe de la transmission et de la réception des paramètres.

Les fonctions VDI sont commandées à travers 5 tableaux d'entrée/sortie :

contrl	**tableau d'entrée/sortie**
intin	**tableau d'entrée**
intout	**tableau de sortie**
ptsin	**tableau d'entrée**
ptsout	**tableu de sortie**

A la place de ptsin et ptsout, AES a les tableaux addrin et addrout. Il existe encore un autre tableau appelé global.

(1) Pour plus de renseignements concernant GEM vous pouvez vous reporter au livre :
"Le livre du GEM sur l'ATARI ST" édité par MICRO APPLICATION

7.2.1 APPELS DE VDI

Les adresses de départ des tableaux VDI se trouvent dans les variables réservées contrl, intin, intout, ptsin et ptsout.
L'instruction POKE permet de remplir les tableaux avec les paramètres d'entrée. Il faut faire attention à cet égard au fait que les tableaux contrl, intin et intout contiennent des valeurs sur deux octets (des mots). Un paramètre désigné dans la littérature sur GEM comme par exemple intin(3) devra donc être fixé avec l'instruction :

POKE INTIN+6,valeur

Il faut absolument s'en souvenir !

Les fonctions VDI sont numérotées. Ce numéro de fonction est stocké, avant l'appel de VDI, dans contrl(0) avec l'instruction :

POKE CONTRL,numéro de fonction

VDI peut ensuite être appelé avec l'instruction :

VDISYS

Après appel et exécution de la fonction, d'éventuels paramètres de retour pourront être lus. Si la fonction VDI renvoie par exemple une valeur dans intout(2), celle-ci pourra être affectée à une variable avec l'instruction :

variable=PEEK(INTOUT)

7.2.2 APPELS D' AES

Comparés aux appels de VDI, les appels d'AES ne sont pas aussi simples à manier. Les adresses de tableaux ne sont en effet pas toutes fournies ici dans des variables réservées.

Il y a seulement la variable GB qui contient l'adresse de la liste d'adresses des tableaux AES. Cela semble assez compliqué alors examinez plutôt l'illustration suivante :

GB ⟶ IIIII ⟶ **contrl**

IIIII ⟶ **global**

IIIII ⟶ **intin**

IIIII ⟶ **intout**

IIIII ⟶ **addrin**

IIIII ⟶ **addrout**

L'adresse de GB indique donc une zone de 6 fois 4 octets. Cette zone contient donc 6 adresses, les adresses des tableaux AES.

Attention ! Les tableaux intin et intout ne sont pas identiques aux tableaux VDI !

C'est pourquoi nous utiliserons d'autres termes dans l'initialisation suivante d'AES.

```
1000 REM INITIALISATION D'AES
1010 A#=GB
1020 GCONTRL = PEEK(A#)
1030 GLOBAL  = PEEK(A#+4)
1040 GINTIN  = PEEK(A#+8)
1050 GINTOUT = PEEK(A#+12)
1060 ADDRIN  = PEEK(A#+16)
1070 ADDROUT = PEEK(A#+20)
1080 RETURN
```

Une fois ce sous-programme appelé, les adresses de tous les tableaux sont disponibles sous forme de variables.

Notez que les tableaux addrin et addrout sont des tableaux 4 octets. addrin(3) sera par exemple fixé avec l'instruction :

POKE ADDRIN+12,valeur

L'appel d'AES s'effectue ainsi :

GEMSYS (numéro de fonction)

Contrairement à ce qui était le cas pour VDI, le numéro de fonction pour AES ne doit pas être placé dans contrl(0) mais il doit être transmis lors de l'appel d'AES. Il n'est donc pas nécessaire de fixer le numéro de fonction dans le tableau contrl pour AES.

Après appel d'AES, vous pouvez lire les paramètres en retour comme nous l'avons décrit pour VDI.

7.3 EXEMPLES VDI

Le programme suivant dessine un rectangle arrondi. Cette opération graphique ne vous est pas fournie par le BASIC !

```
10 REM RECTANGLE ARRONDI
20 COLOR 1,1,1,1,1
30 FULLW 2
40 POKE CONTRL,11 : REM n° de fonction pour op. graphiques
50 POKE CONTRL+2,2 : REM nombre de coordonnees dans PTSIN
60 POKE CONTRL+6,0 : REM nombre des entrees INTIN
70 POKE CONTRL+10,8 : REM marque fonction du rectangle arrondi
80 POKE PTSIN,50 : REM coordonnee X angle superieur gauche
90 POKE PTSIN+2,20 : REM coordonnee Y angle superieur gauche
100 POKE PTSIN+4,150 : REM coordonnee X angle inferieur droit
110 POKE PTSIN+6,100 : REM coordonnee Y angle inferieur droit
120 VDISYS
130 RETURN
```

Le programme que nous vous donnons maintenant en conclusion vous permet de mettre en forme le texte sorti à l'écran. Les types d'écriture suivants sont possibles :

1	**gras**
2	**clair**
4	**cursive**
8	**souligné**
16	**étiré**
32	**normal**

Sont également autorisées toutes combinaisons de ces types d'écriture. Il vous suffit alors d'additionner les codes des types d'écriture que vous voulez combiner. 5 vous donnera par exemple une écriture cursive en gras. Voici donc le programme :

```
10 REM SELECTION DE TYPES D'ECRITURE
20 FULLW 2
30 INPUT"MARQUE DU TYPE D'ECRITURE:";A
40 IF A<1 OR A>63 THEN 30
50 POKE CONTRL,106 : REM NUMERO DE FONCTION
60 POKE CONTRL+2,0 : REM ENTREES DANS INTIN
70 POKE CONTRL+6,1 : REM NOMBRE D'ENTREES INTIN
80 VDISYS
90 END
```

7.4 EXEMPLES AES

Le programme de démonstration suivant utilise les routines AES 73 et 74 qui se chargent respectivement de l'expansion et de la contraction d'une boîte. Cette fonction est souvent utilisée dans les programmes GEM pour impressionner visuellement l'utilisateur. Elle peut donner l'impression par exemple que la boîte arrive en volant du fond de l'écran.

```
10 REM EXPANSION ET CONTRACTION D'UNE BOITE
20 GOSUB 1000
30 FOR I=0 TO 7
40 READ A:POKE GINTIN+2*I,A
50 NEXT I
60 DATA 315,200,5,5,0,0,640,400
70 GEMSYS(73):GEMSYS(74)
80 IF INP(2)<>13 THEN 70
90 END
1000 REM INITIALISATION D'AES
1010 A#=GB
1020 GCONTRL = PEEK(A#)
1030 GLOBAL  = PEEK(A#+4)
1040 GINTIN  = PEEK(A#+8)
1050 GINTOUT = PEEK(A#+12)
1060 ADDRIN  = PEEK(A#+16)
1070 ADDROUT = PEEK(A#+20)
1080 RETURN
```

Le tableau GINTIN se compose de 8 entrées. Les quatre premières entrées fournissent la coordonnée X de l'angle supérieur gauche, la coordonnée Y correspondante, la largeur et la hauteur du (petit) rectangle de base. Les quatre entrées suivantes fournissent sous la même forme les dimensions du rectangle qui en résultera. Le programme se répète chaque fois que vous appuyez sur une touche et il s'interrompt lorsque vous appuyez sur la touche RETURN.

Le second programme d'exemple modifiera la forme de la souris. Les formes suivantes peuvent être entrées :

0	**flèche**
1	**curseur**
2	**abeille**
3	**main avec index pointé**
4	**main plate**
5	**réticule fin**

6	**réticule épais**
7	**réticule en silhouette**

Voici le programme :

```
10 REM FIXER LA FORME DE LA SOURIS
20 FULLW 2
30 INPUT"FORME DE LA SOURIS:";F
40 IF F<0 OR F>7 THEN 30
50 GOSUB 1000
60 POKE GINTIN,F
70 GEMSYS (78)
80 END
1000 REM INITIALISATION D'AES
1010 A#=GB
1020 GCONTRL = PEEK(A#)
1030 GLOBAL  = PEEK(A#+4)
1040 GINTIN  = PEEK(A#+8)
1050 GINTOUT = PEEK(A#+12)
1060 ADDRIN  = PEEK(A#+16)
1070 ADDROUT = PEEK(A#+20)
1080 RETURN
```

Ces exemples ont bien sûr pour seule ambition de vous permettre de mesurer les nombreuses possibilités qu'offre l'utilisation des fonctions GEM à l'intérieur de programmes BASIC. Si vous voulez pousser plus avant l'étude de ce domaine passionnant, nous vous recommandons le "livre du GEM sur l'ATARI ST" (Szczepanowski / Günther) édité par MICRO APPLICATION.

CHAPITRE 8

SOLUTIONS

Solution de la page 31

1)

a) 6C
b) 92
c) BA
d) F0
e) 0C
f) C9

2)

a) 61642
b) 4712
c) 13728
d) 597
e) 61440
f) 2048

3)

a) 183
b) 51
c) 254
d) 21
e) 85
f) 170

4)

a) F730
b) 6000
c) 8001
d) ABCD
e) FFFE
f) 4711

Solutions de la page 54

1. a) valable
   b) valable
   c) AUTO non valable, instruction Atari ST
   d) valable
   e) IF non autorisé car c'est une instruction BASIC
   f) GB non autorisé car c'est une variable du systéme
   g) 4nombre% non autorisé car le nom d'une variable ne doit pas commencer par un chiffre
   h) 255 non autorisé pour la même raison qu'en g)
   i) valable

2.
```
10 INPUT A,B,C,D
20 PRINT A;B
30 PRINT C,D
40 END
```

L'intérêt de l'exercice était de bien se rappeler la différence entre le point-virgule qui provoque une sortie directement l'une derrière l'autre de deux expressions et la virgule qui réalise une tabulation. Notez d'autre part, lorsque vous utilisez ce programme, que vous devez entrer autant de valeurs que l'ordinateur en attend.

3.
```
10 REM ENTREE DE LA HAUTEUR ET
20 REM DE L'HYPOTHENUSE C EN METRES
30 INPUT"ENTREE H,C";hauteur,C
40 A=.5*hauteur*C
50 PRINT"LA SURFACE EST DE ";A;"M^2"
60 END
```

4.
```
10 INPUT"ENTREE TAILLE EN CM";CM
20 REM CALCUL POIDS IDEAL
30 IDEAL=CM-100
40 REM CALCUL 10 POUR CENT
50 POURCENT=IDEAL/100*10
60 IDEAL=IDEAL-POURCENT
70 PRINT"VOTRE POIDS IDEAL EST DE";IDEAL;"KG"
80 END
```

Cette exercice aurait également pu être résolu plus rapidement en réunissant en une seule ligne les lignes 30, 50 et 60 :

```
30 IDEAL=(CM-100)-(CM-100)/100*10
```

Cette façon d'écrire aurait cependant pour gros inconvénient de ne pas permettre à une autre personne qui lirait le programme de comprendre rapidement et clairement de quel calcul il s'agit. Il est cependant vrai que cette façon d'écrire, si elle est moins claire, permet toutefois d'économiser de la place en mémoire. Il faut donc essayer d'arriver en permanence, en fonction de la capacité de mémoire disponible, à un compromis entre les exigences de la clarté (lignes de commentaires avec REM, aération du programme par des lignes sans instructions, une seule instruction par ligne) et la meilleure utilisation de la mémoire disponible. Pour des programmes de moyenne importance, nous ne saurions donc trop vous recommander de ne pas hésiter à présenter les programmes de la façon la plus claire possible.

5.
```
10INPUT"HAUTEUR,LONGUEUR,PROFONDEURENCM";haut,long,prof
20 REM CALCUL DU VOLUME
30 vol=haut*long*prof
40 REM CALCUL DES LITRES
50 vol=vol/1000
60 PRINT"CONTENU DE L'AQUARIUM";vol;"LITRES"
70 END
```

Ce programme entre d'abord les paramètres de hauteur, longueur et de profondeur de l'aquarium en centimètres. Vous voyez qu'il est très pratique de pouvoir donner des noms significatifs aux variables. La ligne 30 calcule ensuite le volume en centimètres cube. La ligne 50 divise le volume ainsi obtenu par 1000 pour obtenir la contenance de l'aquarium en litres.

6.
```
10 INPUT A,B,C,D
20 PRINT"A";A
30 PRINT"B";B
40 PRINT"C";C
50 PRINT"D";D
60 END
```

Nous espérons que vous n'avez fait qu'une bouchée de ces exercices. Si cependant vous avez encore rencontré quelques difficultés avec un exercice ou un autre, relisez les passages des chapitres précédents qui se rapportent au problème posé dans ce ou ces exercices.

Solutions de la page 67

1.
```
10 REM VIDER L'ECRAN
20 CLEARW 2
30 REM GENERER DES NOMBRES ALEATOIRES
40 DE1=INT(6*RND(1))+1
50 DE2=INT(6*RND(1))+1
60 REM SORTIE DU RESULTAT
70 PRINT"DE 1:";DE1,"DE 2:";DE2
80 END
```

Voici à quoi votre programme devrait ressembler.

Les solutions qui vous sont proposées ici ne sont toutefois que des propositions car tous les chemins ménent à Rome et la seule chose qui compte en informatique, c'est le résultat. Si vous lancez plusieurs fois ce programme avec RUN, vous obtiendrez chaque fois des valeurs différentes.
La ligne 20 vide l'écran. Les lignes 40 et 50 affectent aux variables DE1 et DE2 deux nombres aléatoires. Si vous avez eu du mal à définir les bornes inférieure et supérieure des nombres aléatoires, relisez le chapitre sur les nombres aléatoires.

2.
```
10 REM ENTREE DES COTES DU TRIANGLE
20 INPUT"ENTREZ A,B,C EN CM";A,B,C
30 REM CALCUL DE S
40 S=.5*(A+B+C)
50 REM CALCUL DE LA SURFACE
60 F=SQR(S*(S-A)*(S-B)*(S-C))
70 REM SORTIE DE LA SURFACE
80 PRINT"LA SURFACE DU TRIANGLE ";
90 PRINT"EST DE";F;" CM^2"
100 END
```

Faites attention dans ce programme à calculer d'abord la valeur S puisque cette variable est utilisée pour le calcul de la surface. La traduction en BASIC de la formule ne devrait pas vous avoir présenté de difficulté. Notez simplement que vous ne pouvez pas écrire les formules mathématiques sous leur forme classique dans vos programmes. Si vous le faites, vous risquez de provoquer une interruptiondevotreprogrammeetunmessageSYNTAXERROR,chaque fois que l'écriture classique se distingue de l'écriture BASIC.

3.
```
10INPUT"APPUYEZSURUNETOUCHEPUISSURRETURN";A$
20 A=ASC(A$)
30 PRINT"LE CODE ASCII DE ";A$;" EST";A
40 END
```

Si vous avez essayé ce programme et tenté d'obtenir ainsi le code ASCII de la touche RETURN ou de la virgule, vous aurez vu

apparaître un message d'erreur. L'instruction INPUT présente en effet l'inconvénient de n'utiliser la virgule que pour séparer entre elles des données entrées. Si vous appuyez par ailleurs sur la touche RETURN, l'ordinateur considère que vous êtes en train de valider votre entrée et que vous n'avez donc rien entré. C'est la chaîne vide qui est donc affectée à la variable alphanumérique. Comme l'ordinateur ne peut évidemment pas calculer le code ASCII d'une chaîne vide, il vous envoie un message d'erreur. Nous expliquerons plus loin, à propos de l'instruction INP, comment il est possible de pallier cette difficulté.

4.
```
10 G=9.81
20 INPUT"COMBIEN DE SECONDES";T
30 S=.5*G*T^2
40 PRINT"LE CORPS EST TOMBE D'UNE";
50 PRINT" HAUTEUR DE";S;" METRES"
60 FIN
```

La ligne 10 qui affecte à la variable G la valeur 9.81 réalise ce qu'on appelle l'INITIALISATION D'UNE VARIABLE. Ce procédé qui consiste à affecter en début de programme différentes valeurs à des variables permet dans la suite du programme de ne plus avoir à utiliser que les variables au lieu de ces valeurs qui peuvent parfois comporter beaucoup de chiffres. Dans de longs programmes, c'est là un moyen de gagner de la place en mémoire qui ne doit pas être négligé.

5.
```
10 INPUT"CONSOMMATION EN LITRES";lit
20 INPUT"NOMBRE DE KILOMETRES PARCOURUS";km
30 consommation=lit/km*100
40PRINT"CONSOMMATIONAUX100KM";consommation;"LITRES"
50 END
```

Ce programme ne nécessite pas de commentaire particulier.

Si vous avez résolu ces différents problèmes sans difficulté, vous pouvez maintenant passer aux chapitres suivants. Dans le cas contraire, nous vous invitons à relire les passages correspondant aux problèmes que vous avez eu du mal à résoudre.

Solutions de la page 81

1. La réponse a) est la bonne.

2. Vous obtenez l'expression MALIN.

3. Vous obtenez à nouveau l'expression ROTOR.

4. B$=LEFT$(A$,3)+MID$(A$,6,8)+MID$(A$,4,3)+MID$(A$,16,1) + RIGHT$(A$,1)

   Il ne s'agit là que d'une possibilité parmi tant d'autres et vous pouvez vous amuser à réaliser cette chaîne de caractères de diverses façons pour maîtriser parfaitement ces instructions de chaînes.

Solutions de la page 101

1.
```
10 REM ENTREE REVENU ANNUEL
20 INPUT"REVENU ANNUEL EN FRANCS";revenu
30 IF revenu>50000 THEN 70
40 REM CALCUL DE 33 POUR CENT
50 impot=revenu/100*33
60 GOTO 90
70 REM CALCUL DE 51 POUR CENT
```

```
80 impot=revenu/100*51
90 PRINT"IMPOT A PAYER ";
100 PRINT impot;" F"
110 END
```

La ligne 20 demande le revenu annuel. La valeur entrée est affectée à la variable revenu. La ligne 30 teste si le revenu est supérieur à 50 000F. Si ce n'est pas le cas, le programme calcule l'impôt sur la base de 33% du revenu et sort l'impôt calculé. Si par contre le revenu est supérieur à 50 000F, la ligne 80 calcule l'impôt sur la base de 51% du revenu et sort l'impôt calculé.

2. Cet exercice peut être résolu d'au moins deux façons différentes. Voyons tout d'abord une première solution avec l'instruction IF ... THEN :

```
10 REM SOMME DES NOMBRES DE 1 A 100
20 A=A+1
30 S=S+A
40 IF A < 100 THEN 20
50 PRINT"LA SOMME DES NOMBRES DE 1 A 100 =";S
60 END
```

La ligne 20 crée un compteur pour les nombres à additionner de 1 à 100. La ligne 30 calcule la somme des différentes valeurs que prend A: 1+2+3+ ... +100. La ligne 40 effectue une comparaison et la ligne 50 sort enfin la somme S des nombres de 1 à 100.
La seconde solution est tirée de la constatation du fait que nous avons ici affaire à une série arithmétique, c'est-à-dire que la différence entre les différents termes de la série est constante. La somme des différents termes de la série peut donc être calculée d'après la formule :

Sn=n/2(A1+An)

n est le nombre de termes de la série pris en compte, A1 est le

premier terme de la série et An le dernier terme.
La deuxième solution vers laquelle nous nous acheminons ainsi constituerait donc même une solution générale de ce type de problèmes. Voici comment pourrait se présenter notre programme :

```
10 INPUT"Nombre de termes";n
20 INPUT"Premier terme";A1
30 INPUT"Dernier terme"AN
40 REM CALCUL
50 SN=N/2*(A1+AN)
60 REM SORTIE
70 PRINT"La somme est";SN
80 END
```

3.
```
10 REM 6 PARMI 49
20 Z=Z+1
30 L=INT(49*RND(1))+1
40 IF Z > 6 THEN END
50 PRINT L;
60 GOTO 20
```

Pour la première fois, nous n'avons pas placé l'instruction END à la fin (physique) d'un programme. Cette instruction marque en effet la fin logique d'un programme qui ne coïncide plus nécessairement avec la dernière ligne du programme, dès lors qu'on n'est pas dans un programme à déroulement linéaire. La structure très simple de ce programme n'appelle pas d'autre commentaire.

4. Il fallait comprendre dans cet exercice que seules les dernières valeurs de A et Z sont sorties. L'instruction PRINT est en effet placée en dehors de la boucle. La solution est donc :

```
52    9
```

Si vous aviez pensé que la deuxième valeur devait être 8, réfléchissez au fait que le programme saute en ligne 20 tant que Z est inférieur à 9. Ce n'est qu'à partir du moment où Z est égal à 9 que la condition n'est plus remplie et que la programme passe à la sortie en ligne 40.

5.
```
10 REM ENTREE DE CHAINE ET SOUS-CHAINE
20 INPUT"Quelle chaine";A$
30 INPUT"Quelle sous-chaine";B$
40 I=I+1
50 C$=MID$(A$,I,LEN(B$))
60 IF C$=B$ THEN PRINT"TROUVE":END
70 IF I>LEN(A$) THEN PRINT"PAS TROUVE":END
80 GOTO 40
```

Reconnaissons que cet exercice était vraiment difficile puisqu'il s'agissait de simuler l'instruction INSTR. Votre programme ne doit pas nécessairement être identique à celui que nous vous indiquons ici mais il devrait de toute façon contenir l'instruction de création de la chaîne de comparaison car c'était en effet le noeud du problème. La difficulté était en effet d'indiquer à l'ordinateur avec l'instruction MID$ à partir de quel caractère de la chaîne principale, il devait commencer la comparaison avec la sous-chaîne mais aussi sur combien de caractères devait se faire cette comparaison. Comme la sous-chaîne devait être recherchée dans la totalité du texte de la chaîne principale, nous avons donc réalisé avec I un compteur qui nous permet de commencer la comparaison à partir du premier caractère puis de la poursuivre à partir des caractères suivants tant que la sous-chaîne n'a pas été trouvée. Le programme parcourt une boucle de comparaison tant que I est inférieur à la longueur de A$ qui est fournie par LEN(A$). Pour indiquer au programme combien de caractères de A$ doivent être comparés à la sous-chaîne B$ avec l'instruction MID$, nous utilisons encore une fois la fonction LEN(B$) qui fournit ici le nombre de caractères dont se compose B$. La ligne 60 compare la chaîne à recher-

cher B$ à la chaîne provisoire C$. Si les deux chaînes sont identiques, le programme est terminé. La ligne 70 demande si toute la longueur de A$ a déjà été examinée. Dans ce cas, cela veut dire que B$ ne peut plus être trouvé. Voici une description du fonctionnement du programme avec un exemple concret :

Rechercher dans la chaîne de caractères A$="INFORMATIQUE"
la chaîne de caractères B$="FORMAT".
Nombre de caractères de B$=6
Le programme crée donc les sous-chaînes provisoires suivantes :

1. INFORM
2. NFORMA
3. FORMAT

La troisième sous-chaîne ainsi créée est la chaîne recherchée.

Ouf! C'était un sacré morceau! Nous vous invitons quand même à bien étudier ce programme jusqu'à ce que vous soyez sûr de l'avoir compris dans ses moindres détails. Lorsque ce sera le cas, vous pourrez vous attaquer hardiment au chapitre suivant.

Solutions de la page 130

1.
```
10 REM SERIE HARMONIQUE
20 CLEARW 2
30 PRINT "Jusqu'à quelle somme"
40 PRINT:PRINT"faut-il additionner ?"
50 PRINT
60 INPUT S
70 Z=1
80 SH=SH + 1/Z
90 Z=Z+1
100 IF Z = 50 * INT(Z/50) THEN PRINT Z;"ADDITIONS"
110 IF SH < S THEN 80
```

```
120 PRINT"Après" Z "termes la somme est de" SH
```

Les lignes 20 à 60 vident l'écran puis demandent l'entrée de la somme à obtenir. La ligne 70 met le compteur à 1 et la ligne 80 additionne les différents termes de la somme. La ligne 90 augmente alors le compteur de 1. La ligne 100 teste alors si le compteur a atteint un multiple de 50. Si c'est le cas, l'instruction suivant le THEN est exécutée, c'est-à-dire que le nombre d'additions est affiché, comme le demandait l'exercice. La ligne 110 teste si la valeur à obtenir est déjà atteinte. Lorsque la somme voulue a été atteinte, la ligne 120 indique combien de termes ont été nécessaires ainsi que le résultat même de cette addition.

2.
```
10 REM EQUATION DU SECOND DEGRE
20 CLEARW 2
30 PRINT"Entrée des coefficients A,B,C"
40 PRINT
50 INPUT A,B,C
60 IF A=0 THEN 20:REM A DOIT ETRE <> 0
70 D=B*B-4*A*C
80 IF D < 0 THEN 140
90 X1 =(-B+SQR(D))/(2*A)
100 X2 =(-B-SQR(D))/(2*A)
110 PRINT"Solution pour X1 =";X1
120 PRINT"Solution pour X2 =";X2
130 GOTO 150
140 PRINT"Pas de points zéro réels !!"
150 END
```

La traduction du problème en programme ne devrait pas vous avoir causé de difficulté. Il faut noter que si A égale 0, il devra être divisé par 2*A suivant la formule. Il y aurait alors une division par 0, ce qui n'est pas autorisé. C'est pourquoi nous avons exclu ce cas au début du programme (ligne 60).

3. L'écran se vide puis le message "Valeur non-valable." est sorti sur l'écran. Il fallait penser que l'instruction placée à la suite du GOTO sera elle aussi exécutée.

Solutions de la page 157

1.
```
10 REM Entree de noms
20 DIM Y$(6)
30 FOR I=1 TO 6
40 INPUT"Nom";Y$(I)
50 NEXT I
60 REM Premier nom dans l'ordre alphabétique
70 Y$(0)=Y$(1)
80 FOR I=2 TO 6
90 IF Y$(0) <= Y$(I) THEN 110
100 Y$(0)=Y$(I)
110 NEXT I
120 PRINT "Le premier nom est ";Y$(0)
130 END
```

La première partie du programme ne présentait pas de difficulté puisque nous avons utilisé la même technique dans plusieurs exemples. Nous avons pu utiliser ici une boucle FOR ... NEXT puisque nous savions qu'il fallait entrer exactement 6 noms. La deuxième partie était plus complexe et vous pouvez être fier de vous si vous y êtes arrivé également. Nous avions déjà parlé du stockage provisoire de valeurs et de données et c'est cette technique que nous avons utilisée ici. L'important est qu'aucune donnée ne soit perdue, quelle que soit la variable que vous ayez utilisée pour le stockage provisoire des données. Nous avons utilisé l'élément 0 du tableau puisqu'il n'a pas été employé jusqu'ici. La ligne 70 "sauve" donc le contenu de Y$(1) en Y$(0). En ligne 80 commence la boucle FOR ... NEXT avec la valeur de départ 2.

Il n'est en effet pas utile de comparer le premier élément avec lui-même. La ligne 90 compare ensuite les divers éléments du tableau, dans l'ordre où ils ont été entrés. Si le nom qui se trouve en Y$(0) est "déjà" plus petit que celui qui se trouve momentanément en Y$(I), le programme saute à la ligne 110 et la variable-compteur est augmentée de 1.
Si le nom en Y$(0) est par contre plus grand que celui qui se trouve momentanément en Y$(I), la valeur de Y$(I) est affectée à Y$(0). Lorsque la variable-compteur atteint la valeur 6, le nom recherché se trouve en Y$(0). La ligne 110 peut alors le sortir avec un message de commentaire.

2.
```
10 REM Entree de nombres
20 DIM X(6)
30 FOR I=1 TO 6
40 X(I)=INT(100*RND(1))+50
50 NEXT I
60 REM Chercher le plus grand nombre
70 X(0)=X(1)
80 FOR I=2 TO 6
90 IF X(0) <= X(I) THEN 110
100 X(0)=X(I)
110 NEXT I
120 PRINT "Le plus grand nombre est";X(0)
130 END
```

La structure de ce programme est identique à celle de l'exemple précédent. Si vous aviez résolu le premier exercice, vous devez donc avoir aussi résolu le second. Les seules différences entre les deux programmes tiennent à la différence de type de tableau (ici tableau numérique) et à la génération de nombres aléatoires en ligne 40. Les comparaisons pour rechercher le plus grand nombre suivent le même principe que la recherche du premier nom dans l'ordre alphabétique dans l'exemple précédent.

3.
```
10 Formule de la serie
20 DIM X(6)
30 FOR I=1 TO 6
40 X(I)=I*I-I
50 NEXT I
60 REM Sortie du tableau
70 FOR I=1 TO 6
80 PRINT X(I)
90 NEXT I
100 END
```

La structure du programme ne pose pas de problème particulier. Remarquons simplement que vous pouvez avoir trouvé une autre formule pour calculer les valeurs de notre tableau. La seule chose qui compte est que votre programme sorte bien les valeurs que nous vous avions données.

ANNEXE

L'Atari 520 ST était livré au départ avec une carte de référence BASIC provisoire, c'est-à-dire sans le source book BASIC. Cette carte n'était ni complète ni exempte d'erreurs. La liste d'instructions suivante contient toutes les instructions qui n'étaient pas décrites dans la carte de référence ou qui y étaient décrites de façon erronnée.

CLEARW

Syntaxe : CLEARW numéro de fenêtre
Fonction : Vide le contenu d'une fenêtre de l'écran.
numéro de fenêtre doit être compris entre 0 et 3 inclus.

CLOSEW

Syntaxe : CLOSEW numéro de fenêtre
Fonction : Ferme une fenêtre de l'écran, c'est-à-dire que le fenêtre disparaît du desk top.

CONTRL

Fonction : Variable d'adresse réservée pour l'appel de GEM.

EQV

Syntaxe : X=op1 EQV op2
Fonction : Fait subir aux deux opérandes op1 et op2 la négation d'un ou exclusif.

FILL

Syntaxe : FILL x,y
Fonction : remplit la surface indiquée par les coordonnées.

FULLW

Syntaxe : FULLW numéro de fenêtre
Fonction : Fixe les dimensions de l'écran entier comme dimensions d'une fenêtre écran.

GB

Fonction : Variable d'adresse réservée pour l'appel de GEM.

GEMSYS

Syntaxe : GEMSYS (numéro de fonction)
Fonction : Appelle la fonction AES dont le code d'opération est indiqué par numéro de fonction.

GOTOXY

Syntaxe : GOTOXY ligne,colonne
Fonction : Positionne le curseur sur la position de l'écran déterminée par ligne et colonne.

IMP

Syntaxe : X=op1 IMP op2
Fonction : Fait subir aux deux opérandes op1 et op2 une opération d'implication (voir chapitre 1.6).

INKEY$

Syntaxe : A$=INKEY$
Fonction : Lit un caractère au clavier et le stocker dans A$.

ATTENTION ! Ne fonctionne pas dans la version BASIC actuelle !

INTIN

Fonction : Variable d'adresse réservée pour l'appel de GEM.

INOUT

Fonction : Variable d'adresse réservée pour l'appel de GEM.

MOD

Syntaxe : X MOD Y
Fonction : Fournit le reste de la division de X par Y.

OPEN

Syntaxe : OPEN"mode",#numérodefichier,"nomdefichier",longueur d'enregistrement
Fonction : Ouvre un fichier. Important: le mode (I, O ou R) doit être placé entre guillemets. La longueur d'enregistrement ne concerne que les fichiers relatifs (mode R).

OPENW

Syntaxe : OPENW numéro de fenêtre
Fonction : Ouvre une fenêtre de l'écran

PTSIN

Fonction : Variable d'adresse réservée pour l'appel de GEM.

PTSOUT

Fonction : Variable d'adresse réservée pour l'appel de GEM.

SOUND

Syntaxe : SOUND voix,volume,note,octave,durée
Fonction : Commande les générateurs de son (voir chapitre MUSIQUE).

VDISYS

Syntaxe : VDISYS
Fonction : Appelle une fonction VDI. Le numéro de fonction est transmis à travers le tableau contrl.

WAVE

Syntaxe : WAVE regmix,regenv,shape,periode,delay
Fonction : Cette instruction commande la forme d'onde des sons produits avec SOUND.

Achevé d'imprimer en mars 1986
sur les presses de l'imprimerie Laballery
58500 Clamecy
Dépôt légal : mars 1986
N° d'imprimeur : 603035

MICRO APPLICATION MICRO APPLICATION

ENTRETIEN ET REPARATION DU VC 1541
Plus de 250 pages.

Ce livre vous permettra de réparer et d'entretenir le lecteur de disquettes Commodore VC 1541. Vous y trouverez amplement décrits les principes mécaniques et électroniques de ce périphérique. Pour ceux qui ne connaissent pas l'électronique cet ouvrage en est aussi un parfait apprentissage.
Courant Octobre.

Réf. : ML134
Prix : 149 FF

LIVRES DIVERS

ATARI ST TRUCS et ASTUCES
Plus de 250 pages.

Un recueil complet de trucs et d'astuces que votre tout nouveau ATARI ST va beaucoup apprécier ! Des graphismes fantastiques à partir de programmes en BASIC, des exemples et des conseils pour programmer en langage C et en Assembleur...
Disponible courant Novembre.

Réf. : ML140
Prix : 149 FF

La BIBLE de l'ATARI ST
Plus de 500 pages.

Ce livre contient un ensemble complet d'informations sur l'ATARI ST, la description HARDWARE de la machine ainsi que des schémas détaillés et amplement expliqués notamment des interfaces V24, du port d'extension, de l'interface midi, la structure des graphiques, du BIOS, de GEM, les adresses systèmes importantes, le fonctionnement de la souris....
Disponible courant Novembre.

Réf. : ML142
Prix : 249 FF

LE LIVRE DU LANGAGE MACHINE DE l'ATARI ST
Plus de 250 pages.

Tout ce qu'il faut savoir pour tirer au mieux parti de votre ATARI ST : système de calcul et de bit manipulation du 68000 , utilisation des registres, structure des commandes, programmation structurée : récursion, piles, procédures et fonctions, listings sources de programmes assembleurs, routines systèmes... Un super livre ! *Disponible Octobre.*

Réf. : ML141
Prix : 149 FF

MICRO APPLICATION MICRO APPLICATION

MICRO APPLICATION MICRO APPLICATION

LE LIVRE DU GEM SUR ATARI ST
Plus de 350 pages.

Cet ouvrage contient ce qu'il est nécessaire de savoir pour utiliser GEM efficacement : Fonctionnement de la souris, Virtual Device Interface, Application Environment Services, Graphics Device Operating System. Description de routines utilisant GEM en BASIC, C et en Assembleur. Utilisation standard et spécifique de l'operating system.

Réf. : ML139
Prix : 149 FF

APPLE 2 TRUCS ET ASTUCES POUR APPLE 2E, 2+, 2C

Très important : l'ouvrage APPLE 2 TRUCS ET ASTUCES repose sur l'expérience acquise dans le travail avec le 2+, 2E et le nouveau super compact 2C. Les PEEKS et POKES intéressants, les bases de la programmation en assembleur, le graphisme couleur, la structure des masques-écran ne sont que quelques-uns des nombreux thèmes abordés. Un aperçu sur l'utilisation de logiciels écrits pour l'APPLE 2 complète ce nouvel ouvrage que tout possesseur d'APPLE 2 doit se procurer.

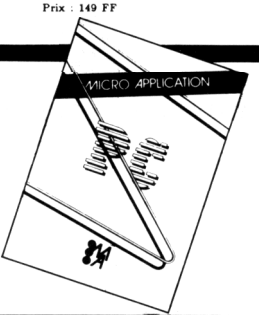
Réf. : ML111
Prix : 149 FF

TRUCS ET ASTUCES POUR TURBO PASCAL

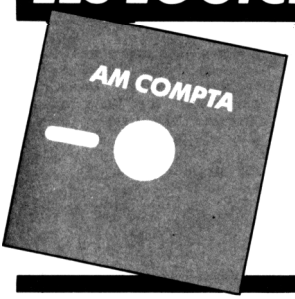
Plus de 250 pages.

Ce livre est un recueil de trucs et d'atuces pour utiliser au mieux Turbo Pascal. Vous trouverez en outre les nombreux conseils et méthodes des programmes exemples et utilitaires comme des procédures de tri, de gestion d'écran ... Un ouvrage très complet indispensable à tout utilisateur de Turbo Pascal.

Réf. : ML133
Prix : 149 FF



LES LOGICIELS AMSTRAD



AM COMPTA
pour CPC 664 et 6128 ou 464+DDI

Ce logiciel sur disquette permet au particulier et au professionnel de tenir sa comptabilité à partir de la saisie des recettes et des dépenses. Celles-ci sont imputées dans des postes comptables et ventilées. La consultation permet la visualisation des codes de trésorerie en dépenses / recettes et soldes, des postes comptables en camembert et histogrammes, la recherche à partir du montant ou du numéro de chèque d'une fiche comptable et la consultation des montants non rapprochés. Edition du journal par postes, du grand livre, ventilation par postes et liste des postes.

Réf. : AM307
PRIX : 750 FF

MICRO APPLICATION MICRO APPLICATION

MICRO
INFO
123
X82
ATARI
80 pages de trucs et astuces, programmes, dossiers pour Amstrad CPC, Commodore Atari ST ...
20 f

MICRO APPLICATION vous présente MICRO INFO, nouveau journal avec des dossiers, des bidouilles, des trucs et astuces, des nouveautés, des programmes et plein de rubriques sympas! (88 pages)
Chaque numéro traite principalement de 3 matériels:

AMSTRAD - COMMODORE - ATARI

carte d'abonnement

Je désire m'abonner à MICRO INFO

☐ Le numéro 1 : 15 F + 5 F pour frais d'envoi
☐ Le numéro 2 : 20 F + 5 F pour frais d'envoi
☐ Les numéros 1 et 2 : 35 F + 5 F pour frais d'envoi
☐ Je choisis de m'abonner pour 4 numéros au prix de 70F

Je règle par ☐ chèque
☐ mandat
☐ CCP

Nom : _____ Prénom : _____
Adresse : _____
Code postal : _____ date et signature :

Veuillez nous retourner cette carte sous pli ainsi que votre règlement à l'adresse suivante :

MICRO APPLICATION
13 rue Sainte Cécile 75009 PARIS

IMPRIMÉ EN FRANCE

Voici un livre qui passionnera tous les utilisateurs de l'ATARI ST désirant connaître à fond le ST BASIC et voulant approfondir ou concrétiser leurs connaissances en créant des applications BASIC complexes aussi bien dans les domaines de la musique, des graphismes que dans les transmissions de paramètres à des routines GEM.

Aperçu du contenu :

- **Les Bases de la programmation**
- **Introduction à la programmation en BASIC**
- **Structures de programmes complexes**
- **Les Tableaux, les sous-programmes, la technique des menus, procédure de tri**
- **Travail avec le lecteur de disquette**
- **La Musique**
- **Le Graphisme**
- **Fonctions GEM sous BASIC**
- **Exemples VDI**
- **Exemples AES**

Réf. : ML 154
Prix : 149 FF

ISBN : 2-86899-045-8